PATENT
P57009

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

BYOUNG-CHUL KIM *et al.*

Serial No.:	*to be assigned*	Examiner:	*to be assigned*
Filed:	20 January 2004	Art Unit:	*to be assigned*

For: DISTRIBUTED ROUTER FOR DYNAMICALLY MANAGING FORWARDING INFORMATION AND METHOD THEREOF

CLAIM OF PRIORITY UNDER 35 U.S.C. §119

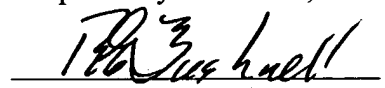
Mail Stop : Patent Application
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

The benefit of the filing date of the following prior foreign application, Korean Priority No.2003-6435 (filed in Korea on 30 January 2003), and filed in the U.S. Patent and Trademark Office on 20 January 2004 is hereby requested and the right of priority provided in 35 U.S.C. §119 is hereby claimed.

In support of this claim, filed herewith is certified copies of said original foreign applications.

Respectfully submitted,



Robert E. Bushnell
Reg. No.: 27,774
Attorney for the Applicant

1522 "K" Street, N.W., Suite 300
Washington, D.C. 20005
(202) 408-9040
Folio: P57009
Date: 1/20/04
I.D.: REB/rfc

대 한 민 국 특 허 청
KOREAN INTELLECTUAL PROPERTY OFFICE

별첨 사본은 아래 출원의 원본과 동일함을 증명함.

This is to certify that the following application annexed hereto is a true copy from the records of the Korean Intellectual Property Office.

출 원 번 호 : 10-2003-0006435
Application Number

출 원 년 월 일 : 2003년 01월 30일
Date of Application JAN 30, 2003

출 원 인 : 삼성전자주식회사
Applicant(s) SAMSUNG ELECTRONICS CO., LTD.

2003 년 06 월 10 일

특 허 청
COMMISSIONER

【서지사항】

【서류명】	특허출원서
【권리구분】	특허
【수신처】	특허청장
【참조번호】	0010
【제출일자】	2003.01.30
【국제특허분류】	H04L
【발명의 명칭】	포워딩정보를 동적으로 관리하는 분산구조라우터 및 그 방법
【발명의 영문명칭】	DISTRIBUTED ROUTER AND METHOD FOR DYNAMICALLY MANAGING FORWARDING INFORMATION
【출원인】	
【명칭】	삼성전자 주식회사
【출원인코드】	1-1998-104271-3
【대리인】	
【성명】	이건주
【대리인코드】	9-1998-000339-8
【포괄위임등록번호】	2003-001449-1
【발명자】	
【성명의 국문표기】	김병철
【성명의 영문표기】	KIM,Byoung Chul
【주민등록번호】	711025-1067017
【우편번호】	449-840
【주소】	경기도 용인시 수지읍 동성1차아파트 101동 103호
【국적】	KR
【발명자】	
【성명의 국문표기】	최병구
【성명의 영문표기】	CHOE,Byung Gu
【주민등록번호】	570226-1140316
【우편번호】	150-093
【주소】	서울특별시 영등포구 문래동3가 54번지 문래엘지빌리지 117동 1501호
【국적】	KR

1020030006435 출력 일자: 2003/6/11

【취지】 특허법 제42조의 규정에 의하여 위와 같이 출원합니다. 대리인 이건주 (인)

【수수료】

【기본출원료】	20 면	29,000 원
【가산출원료】	34 면	34,000 원
【우선권주장료】	0 건	0 원
【심사청구료】	0 항	0 원
【합계】	63,000 원	

【요약서】

【요약】

본 발명은 분산구조라우터의 모든 노드들에서 수집되는 라우팅정보들을 각 노드들이 실시간으로 공유하고 그 라우팅정보들을 이용하여 포워딩정보를 동적으로 관리함으로써, 라우팅정보를 공유하기 위한 각 노드들간의 패킷 포워딩 과정을 생략할 수 있다는 효과가 있다. 그리고 그 노드 정보들을 포워딩테이블에 선택적으로 업데이트 함으로써 노드별 포워딩테이블을 효율적으로 관리할 수 있다는 장점이 있다. 또한 본 발명은 2진 어그리게이션 트리(aggregation tree)를 이용하여 각 노드들의 포워딩정보를 관리하고, 라우팅정보에 해당되는 노드정보들을 어그리게이션하는 가상노드의 레벨을 가변적으로 설정함으로써 분산구조라우터의 노드별 포워딩테이블의 크기를 줄일 수 있다는 장점이 있다.

【대표도】

도 5

【색인어】

분산구조라우터, 포워딩테이블, 어그리게이션

【명세서】

【발명의 명칭】

포워딩정보를 동적으로 관리하는 분산구조라우터 및 그 방법{DISTRIBUTED ROUTER AND METHOD FOR DYNAMICALLY MANAGING FORWARDING INFORMATION}

【도면의 간단한 설명】

도 1은 분산구조라우터에 대한 예시도,

도 2는 분산구조라우터의 통상적인 프로세스 구조를 도시한 도면,

도 3은 본 발명의 일 실시 예에 따른 분산구조라우터의 프로세스 구조를 도시한 도면,

도 4a는 본 발명의 일 실시 예에 따라 포워딩정보를 관리하기 위해 생성된 어그리게이션 트리(aggregation tree)의 각 노드별 관리데이터 구조에 대한 예시도,

도 4b는 본 발명의 일 실시 예에 따라 포워딩정보를 관리하기 위해 생성된 어그리게이션 트리(aggregation tree) 구조에 대한 예시도,

도 4c는 포워딩주소정보(prefix)의 길이별 분포를 나타낸 그래프,

도 5는 새롭게 삽입된 라우팅정보를 본 발명의 일 실시 예에 따라 관리하는 방법에 대한 처리 흐름도,

도 6a 내지 도 6c는 포워딩정보를 삽입하는 과정에 대한 처리 흐름도,

도 7은 삭제된 라우팅정보를 본 발명의 일 실시 예에 따라 관리하는 방법에 대한 처리 흐름도,

도 8a 내지 도 8g는 본 발명의 일 실시 예에 따른 포워딩정보를 동적으로 관리하는 방법에 대한 알고리즘 예를 나타낸 도면,

도 9a 및 도 9b는 새롭게 삽입된 포워딩정보를 관리하는 방법을 도식화하여 설명한 도면,

도 10a 및 도 10b는 삭제된 포워딩정보를 관리하는 방법을 도식화하여 설명한 도면,

도 11a 내지 도 11d는 본 발명의 일 실시 예에 따른 포워딩정보 동적관리 방법의 효과를 입증하기 위한 테스트 결과들.

【발명의 상세한 설명】

【발명의 목적】

【발명이 속하는 기술분야 및 그 분야의 종래기술】

<14> 본 발명은 분산구조라우터에서 포워딩정보를 관리하는 방법에 관한 것으로서, 특히, 분산구조라우터의 모든 노드들이 각 노드들에서 수집되는 라우팅정보들을 실시간으로 공유하고 그 라우팅정보에 의거하여 포워딩정보를 어그리게이션(aggregation)에 의해 동적으로 관리하는 방법에 관한 것이다.

<15> 대용량 초고속 네트워크의 발전에 따라 라우터는 기존의 중앙집중형 구조에서 분산처리형의 새로운 구조로 변화해가고 있다.

<16> 중앙집중형 라우터는 중앙의 프로세서에서 라우팅 프로토콜을 구현하고, 그

라우팅 프로토콜이 수집한 라우팅정보들을 상기 중앙의 프로세서가 관리하였다.

예컨대, 중앙집중형 라우터에서는 중앙의 프로세서가 라우팅 테이블을 계산하고, 각각의 라인카드에 라우팅 테이블을 분배해주는 역할을 하였다. 따라서, 라인 카드에 의한 패킷 포워딩이 중앙의 라우트 프로세서로부터 전해지는 라우팅 테이블 정보에 의해서 이루어졌다.

<17> 반면, 분산처리형 라우터는 이와 같이 중앙의 프로세서에 집중된 작업들을 다수개의 프로세서에 의해 분산처리 하도록 한다. 따라서, 분산처리형 라우터는 중앙집중형 라우터에 비해 대용량의 데이터를 처리할 수 있다. 예를 들어, 분산처리형 라우터에서는 라우팅 프로토콜을 관리하는 프로세서와 라우팅테이블을 계산하는 프로세서와 패킷 포워딩을 관리하는 프로세서들을 각각 따로 두고, 각각의 프로세서에서 해당 작업들을 분산처리함으로써 라우팅 성능을 향상시켰다.

<18> 도 1은 분산구조라우터에 대한 예시도이다. 특히, 도 1은 갤럭시 아이.피. 라우터(Galaxy IP Router)의 예를 나타내었다.

<19> 도 1을 참조하면, 분산구조라우터(100)는 다수개의 라우팅 노드들(RNs: Routing Nodes)(110, 120, 130, 140)을 포함하고, 그 RNs(110, 120, 130, 140)는 스위칭 모듈(SWM: switching module)(150)에 의해 서로 연결된다. 한편, 각각의 RN(110, 120, 130, 140 중 어느 하나)에는 입출력 프로세서(IOP: Input Output Processor)(111)가 탑재되고, 그 IOP(111)는 두 개의 물리적인 매체들(PMD-A(Physical Medium Device A)(112) 또는 PMD-B(113))로부터 패킷을 전달받도록 설계되었다(designed).

<20> 이러한 RNs(110, 120, 130, 140)는 각각 대응되는 서브-넷(sub-network)을 지원하기 위한 고유의 라우팅 테이블(own routing table)과 그 라우팅 경로를 처리하기 위한

고유의 프로세서들(own processors)을 가진다. 그리고, 모든 RNs(110, 120, 130, 140)는 이를 통해 독자적인 라우팅 프로토콜을 실행시키고(running), 독자적으로 포워딩(forwarding)을 수행한다. 하지만, 이러한 RNs(110, 120, 130, 140)는 사용자의 관점에서는 하나의 라우터로 인식되어진다. 이를 위해, RNs(110, 120, 130, 140)는 SWM(150)을 통해 연결된 다른 RNs(110, 120, 130, 140)의 라우팅 테이블들을 통합적으로(globally) 관리한다. 이 때, 각 RN(110, 120, 130, 140 중 어느 하나)에 실제로 연결된 물리적인 서브-넷(sub-network)을 로컬영역(Local area)(B)이라 하고, SWM(150)을 통해 RNs(110, 120, 130, 140)가 연결되어 형성된 네트워크를 가상영역(Virtual area)(A)이라 한다.

<21> 도 2는 분산구조라우터의 통상적인 프로세스 구조를 도시한 도면이다. 특히, 도 2에서는 분산구조라우터(100)가 4개의 RNs를 갖는 경우에 각 RN에 탑재된 IOP의 프로세스 구조 및 그 IOP들(IOP#1(10), IOP#2(20), IOP#3(30), IOP#4(40))이 SWM(50)을 통해 연결된 상태를 도시하였다.

<22> 도 2를 참조하면, IOP#1(10)은 다수개의 라우팅 프로토콜들(ripd(11), ospfd(12), bgpd(13), isisd(14))과, IOP 관리 프로세서(glued: Galaxy Loosely Unified Environment Daemon)(15), 라우팅테이블(routing table)(16), 포워딩테이블(forwarding table)(17)을 포함한다.

<23> 라우팅 프로토콜들(ripd(11), ospfd(12), bgpd(13), isisd(14))은 각각 고유의 수집기준에 의해 라우팅 정보들을 수집한다. 라우팅테이블(16)은 라우팅 프로토콜들(ripd(11), ospfd(12), bgpd(13), isisd(14))에서 수집된 라우팅 정보들을 저장한다.

포워딩테이블(17)은 라우팅테이블(16)에 저장된 라우팅 정보들을 계산하여 얻어진 포워딩정보를 저장한다.

<24> IOP 관리 프로세서(glued)(15)는 라우팅 프로토콜들(ripd(11), ospfd(12), bgpd(13), isisd(14))에서 수집된 라우팅 정보들을 라우팅테이블(16)에 저장하고, 그 라우팅 정보들을 계산하여 얻어진 포워딩정보를 포워딩테이블(17)에 저장하는 일련의 과정들을 수행하고 관리한다. 또한, IOP 관리 프로세서(glued)(15)는 이 때 얻어진 라우팅테이블(16)을 SWM(50)을 통해 다른 IOP들(IOP#2(20), IOP#3(30), IOP#4(40))에게 통지(advertise)한다.

<25> 한편, IOP#1(10)은 라우팅 프로토콜들(ripd(11), ospfd(12), bgpd(13), isisd(14)), IOP 관리 프로세서(glued)(15) 및 라우팅테이블(16)을 포함하는 시스템 프로세서(system processor)영역과, 포워딩테이블(17)을 포함하는 네트워크 프로세서(network processor)영역으로 구분된다. 시스템 프로세서(system processor) 영역은 라우팅정보의 수집, 라우팅경로 계산에 의한 포워딩테이블 관리 및 타 IOP들(IOP#2(20), IOP#3(30), IOP#4(40))과의 라우팅테이블 공유를 위한 일련의 처리 과정들을 수행하고, 네트워크 프로세서(network processor)는 포워딩테이블(17)에 의거하여 로컬영역(local area)에 연결된 네트웍장비들간의 포워딩을 수행한다. 이로 인해, 분산구조라우터가 대용량의 데이터를 보다 신속하게 처리할 수 있게 되는 것이다.

<26> 도 1 및 도 2에 예시된 것처럼 분산구조라우터(100)에서 대용량의 데이터를 보다 신속하게 처리하게 하기 위해서는 분산구조라우터(100)에 포함된 각 RN에서 개별 관리하는 포워딩테이블(forwarding table)들을 다른 RNs에서도 알 수 있도록 하는 것이 필수적이다. 이를 위해, 종래의 분산구조라우터(100)에서는 각 RN에서 개별 관리하는 포워딩

테이블들을 SWM(150)을 통해 상호 전송함으로써 각 RN에서 모든 RNs의 포워딩테이블을 통합적으로(globally) 관리하도록 하고 있다. 예를 들어, 분산구조라우터(100)에 10개의 RNs가 있고 그 각각의 RN이 고유한 10,000개의 포워딩정보들(forwarding entries)을 가진다면, 각 RN은 100,000(=10,000*10(RNs))의 포워딩정보들을 관리하여야 한다. 따라서, 종래의 분산구조라우터(100)들은 이러한 포워딩테이블을 저장하기 위한 대용량의 저장공간을 필요로 했으며, 패킷 포워딩에 오버헤드가 발생한다는 문제점이 있었다.

【발명이 이루고자 하는 기술적 과제】

<27> 본 발명은 이러한 종래의 문제점을 보완하기 위해 안출된 것으로서, 본 발명의 제1 목적은 분산구조라우터에서 각 노드들이 라우팅정보를 공유하기 위해 패킷을 포워딩 함으로써 발생하는 내부 트래픽 증가를 방지할 수 있는 분산구조라우터를 제공함에 있다.

<28> 본 발명의 제2 목적은 분산구조라우터의 모든 노드들에서 수집되는 라우팅정보들을 각 노드들이 실시간으로 공유하는 분산구조라우터를 제공함에 있다.

<29> 본 발명의 제3 목적은 상기 분산구조라우터의 노드별 포워딩테이블의 크기를 줄이기 위한 포워딩정보 관리방법을 제공함에 있다.

<30> 본 발명의 제4 목적은 분산구조라우터의 모든 노드들에서 수집되는 라우팅정보들을 각 노드들이 실시간으로 공유하는 분산구조라우터의 노드별 포워딩정보 관리방법을 제공함에 있다.

<31> 본 발명의 제5 목적은 상기 분산구조라우터의 라우팅정보에 대응되는 노드들 및 그 라우팅정보를 어그리게이션하는 가상노드들로 구성된 2진 어그리게이션 트리

(aggregation tree)를 이용하여 각 노드들의 포워딩정보를 관리하는 방법을 제공함에 있다.

<32> 본 발명의 제6 목적은 상기 어그리게이션하는 가상노드의 레벨을 가변적으로 설정함으로써 포워딩정보를 효율적으로 관리하는 포워딩정보 관리방법을 제공함에 있다.

【발명의 구성 및 작용】

<33> 상기 목적들을 달성하기 위해 본 발명에서 제공하는 분산구조라우터는 상기 라우팅노드들에 각각 포함된 다수의 라우팅 프로토콜에 대응되는 라우팅 프로토콜을 포함하여 상기 라우팅노드들에서 수집된 라우팅정보들을 실시간으로 공유하는 스위칭모듈을 포함하는 것을 특징으로 한다.

<34> 한편, 상기 목적들을 달성하기 위해 본 발명에서 제공하는 상기 분산구조라우터의 포워딩정보 관리방법은 라우팅테이블에 새로운 라우팅정보가 삽입되면 그 라우팅정보에 대응되는 삽입노드의 위치를 탐색하는 제1 과정과, 기 설정된 어그리게이션 가능한 최대 레벨 범위 내에서 삽입노드의 조상노드를 탐색하는 제2 과정과, 상기 제2 과정의 탐색결과 상기 범위 내에 삽입노드의 조상노드가 존재하는 경우 그 조상노드가 포워딩테이블에 업데이트 되었고 삽입노드와 조상노드의 생성영역이 동일하면 그 삽입노드 정보를 포워딩테이블에 업데이트하지 않는 제3 과정과, 상기 제2 과정의 탐색결과 상기 범위 내에 삽입노드의 조상노드가 존재하지 않는 경우 기 설정된 어그리게이션 가능한 최대레벨 범위 내에서 어그리게이션 가능레벨을 재설정한 후 그 재설정된 어그리게이션 가능레벨에 삽입노드를 대표하는 가상노드를 삽입하는 제4 과정과, 상기 라우팅테이블에 삽입된 라

우팅정보의 생성영역을 확인하여 그 생성영역이 가상영역이면 상기 가상노드 정보를 포워딩테이블에 삽입하고, 그 생성영역이 로컬영역이면 상기 삽입노드 정보를 포워딩테이블에 삽입하는 제5 과정을 포함하는 것을 특징으로 한다.

<35> 또한, 상기 목적들을 달성하기 위해 본 발명에서 제공하는 상기 분산구조라우터의 포워딩정보 관리방법은 라우팅테이블에 기 저장된 라우팅정보가 삭제되면 그 라우팅정보에 대응되는 삭제노드의 위치를 탐색하는 제1 과정과, 상기 삭제노드가 포워딩테이블에 업데이트된 경우 어그리게이션 가능한 최대레벨 범위 내에서 상기 삭제노드의 후손노드를 탐색하는 제2 과정과, 상기 제2 과정의 탐색결과 상기 범위 내에 상기 삭제노드의 후손노드가 있으면 후손노드를 가상노드의 새로운 소유자로 변경하고, 그렇지 않은 경우 포워딩테이블에서 상기 삭제노드를 삭제하는 제3 과정을 포함하는 것을 특징으로 한다.

<36> 이하, 본 발명의 바람직한 실시 예들을 첨부한 도면을 참조하여 상세히 설명한다. 도면들 중 동일한 구성요소들은 가능한 한 어느 곳에서든지 동일한 부호들로 나타내고 있음에 유의해야 한다. 또한 본 발명의 요지를 불필요하게 흐릴 수 있는 공지 기능 및 구성에 대한 상세한 설명은 생략한다.

<37> 도 3은 본 발명의 일 실시 예에 따른 분산구조라우터의 프로세스 구조를 도시한 도면이다. 일반적으로 분산구조라우터에는 다수개의 IOP들을 포함하고, 하나의 스위칭 모듈(SWM)에 의해 그 IOP들간에 정보의 교환이 이루어진다. 도 3에서는 이러한 구성을 간략화하여 도시하였다. 즉 도 3의 예에서는 하나의 IOP(IOP#1)(210)와 스위칭모듈(SWM)(250)만을 도시하였다.

<38> 도 3을 참조하면, 본 발명의 일 실시 예에 따른 분산구조라우터의 프로세스 구조는 IOP#1(210)의 시스템 영역에 어그리게이션 트리(aggregation tree)(218)를 더 포함한다.

어그리게이션 트리(aggregation tree)(218)는 본 특허의 출원인에 의해 출원된 특허(출원번호: 10-2002-0075701)에 설명된 바와 같이 분산구조라우터의 각 IOP들이 관리하는 포워딩/라우팅정보들에 대응되는 노드와 그 포워딩/라우팅정보들을 어그리게이션(aggregation)하는 가상노드(delegation node)들을 포함한다.

<39> 또한, 도 3을 참조하면, 본 발명의 일 실시 예에 따른 분산구조라우터는 스위칭모듈(SWM)(250)에 라우팅 프로토콜들(ripd(251), ospfd(252), bgpd(253), Dglued(254))을 포함하고, 이들 라우팅 프로토콜들(ripd(251), ospfd(252), bgpd(253), Dglued(254))은 대응되는 IOP내의 라우팅 프로토콜들(ripd(211), ospfd(212), bgpd(213), glued(214))과 내부적으로 연결(internal peer connection)된다.

<40> 따라서 스위칭모듈(SWM)(250)은 이러한 연결(internal peer connection)에 의해 각 IOP의 라우팅 프로토콜별로 수집된 라우팅정보들을 실시간으로 공유하고, 그 라우팅정보들을 다른 IOP들에게 전달한다. 결과적으로 본 발명의 일 실시 예에 따른 분산구조라우터에서는 이러한 라우팅 프로토콜들간 연결(internal peer connection)에 의해 모든 IOP들이 동일한 라우팅정보들을 공유할 수가 있는 것이다.

<41> 도 3의 예에서, IOP#1(210)과 스위칭 모듈(SWM)(250)에 포함된 RTM(Routing Table Manage)모듈들(215 및 255)은 각 라우팅 프로토콜들(ripd(211), ospfd(212), bgpd(213) 및 glued(214) 또는 ripd(251), ospfd(252), bgpd(253) 및 Dglued(254))에서 얻어진 라우팅정보들을 통합관리하며 그 라우팅정보들의 우선순위를 관리하는 기능을 수행한다.

<42> 도 4a는 본 발명의 일 실시 예에 따라 포워딩정보를 관리하기 위해 생성된 어그리게이션 트리(aggregation tree)의 각 노드별 관리데이터 구조에 대한 예시도이다. 도 4a를 참조하면, 본 발명의 일 실시 예에 따라 포워딩정보를 관리하기 위한 어그리게이션

트리(aggregation tree)의 각 노드들은 해당 포워딩정보가 포워딩될 주소정보(PREFIX), 그 포워딩될 주소의 길이정보(LENGTH), 그 노드를 어그리게이션할 수 있는 레벨(AGGREGATION_LEVEL), 대응되는 노드가 가상노드(delegation node)인 경우 그 노드의 소유자(예컨대, 가상노드를 생성한 노드)정보(INDEX), 포워딩정보의 타입정보(TYPE), 포워딩정보를 생성한 라우팅노드 정보(SOURCE IOP), 해당 포워딩정보를 로컬 포워딩테이블에 저장했는지 여부를 알리기 위한 플래그(FT FLAG) 및 일반 2진 트리의 링크정보(예컨대, 부모노드(parent node) 정보(PARENT), 좌측 서브 트리 정보(L_SUBTREE), 우측 서브트리 정보(R_SUBTREE))를 포함한다. 이러한 노드별 관리데이터들은 어그리게이션 트리(aggregation tree)가 업데이트될 경우 같이 업데이트하여 관리한다.

<43> 해당 노드를 어그리게이션할 수 있는 레벨(aggregation-level)은 운영자가 초기에 최대값(defaultlevel)으로 설정한 후 라우터의 운영에 따라 그 최대값(defaultlevel) 내에서 가변적으로 운영하는 것이 가능하다. 즉, 네트워크 상태에 따라 어그리게이션할 수 있는 레벨을 동적으로 재설정할 수 있으며 이와 같이 어그리게이션할 수 있는 새로운 레벨(dynamiclevel)이 재설정되면 그 새로운 레벨(dynamiclevel) 내에서 라우팅정보를 어그리게이션한 가상노드(delegation)를 생성할 수가 있는 것이다.

<44> 가상노드의 소유자 정보(Index)는 대응되는 노드가 가상노드(delegation node)인 경우 그 노드를 생성한 노드 정보를 저장한다. 가상노드의 소유자정보(Index)는 가상노드(delegation)로부터 그 노드를 생성한 노드정보를 찾아갈 수 있는 정보를 저장한다. 예를 들어, 가상노드의 좌측 자노드의 우측 자노드가 대응되는 가상노드를 생성한 경우, 이진트리에서는 일반적으로 좌측 자노드는 '0'값을 우측 자노드는 '1'값을 가지게 되므로, 그 가상노드의 소유자정보(Index)는 '01'이 된다.

<45> 포워딩정보의 타입정보(Type)는 IOP 내에서 해당 포워딩정보를 생성한 라우팅 프로토콜의 종류(예컨대, BGP, SDPF, RIP 등)를 말한다. 가상노드는 실제 포워딩정보와 구별하기 위한 별도의 타입(Delegation)을 지정하여 사용하며, 다른 타입(예컨대, BGP, SDPF, RIP 등)의 포워딩정보에 대해서 가장 낮은 우선순위를 가진다. 이는 가상노드는 가상의 포워딩정보이므로 라우팅 프로토콜들이 제공하는 실제의 포워딩정보들에게 높은 우선순위를 부여하기 위함이다.

<46> 이처럼 포워딩정보의 타입정보를 설정하기 위해서 라우팅노드들은 포워딩정보의 생성영역을 확인한 후 로컬영역에서 생성된 포워딩정보는 그 포워딩정보를 생성한 프로세서의 종류에 의거하여 포워딩정보의 타입을 설정하고, 가상영역에서 생성된 포워딩정보는 가상의 타입을 설정하여 그 타입을 포워딩정보의 타입정보(type)에 저장한다. 따라서 라우팅노드들은 포워딩정보의 주소정보를 분석하여 그 분석결과 포워딩정보의 주소가 사설 IP(internet protocol)주소이면 대응되는 포워딩정보가 가상영역으로부터 전달된 것으로 판단하고, 그렇지 않으면 대응되는 포워딩정보가 로컬영역에서 생성된 것으로 판단한다.

<47> 이 때, 사설 IP(internet protocol) 주소란 일반적으로 소정의 지역네트워크에서만 가용한 주소를 말하는 것으로서, 그 지역네트워크 내에 연결된 노드들을 식별하기 위한 주소정보이다. 따라서 그 사설 IP는 해당 지역네트워크 외부에서는 사용할 수 없다. 또한 일반적으로 분산구조라우터는 그 분산구조라우터에 포함된 각 라우팅노드들을 식별하기 위해 각 라우팅노드들에게 사설 IP 주소를 부여한다. 따라서 본 발명에서는 이러한 사설 IP 주소를 이용하여 포워딩정보의 생성영역을 확인하는 것이다.

<48> 도 4b는 본 발명의 일 실시 예에 따라 포워딩정보를 관리하기 위해 생성된 어그리게이션 트리(aggregation tree) 구조에 대한 예시도이다. 이와 같이 본 발명을 수행하기 위한 어그리게이션 트리(aggregation tree)는 2진 트리 형태를 가지며, 라우팅정보들의 삽입 및 삭제에 따른 어그리게이션 트리(aggregation tree)의 변동 정보들은 도 9a, 도 9b, 도 10a 및 도 10b를 참조한 설명시 구체적으로 설명한다.

<49> 본 발명에서 어그리게이션(aggregation)을 위해 가상노드(delegation)를 만드는데 레벨에 제한값(default level)을 설정하는 것은 본 발명의 성능 검증에 사용될 real BGP core 라우팅 테이블의 라우팅 엔트리 분포가 도 4c와 같이 노드의 프리픽스(prefix) 길이(length) 15 ~ 24인 지점에서 집중적으로 분포한다. 따라서 이러한 분석결과로부터 동작으로 포워딩정보의 어그리게이션 및 삭제(retrieve)시에 알고리즘의 오버헤드를 줄일 수 있기 때문이다.

<50> 본 발명에 따르면 도 1에서 설명된 Galaxy router가 멀티-랙(multi-rack)으로 확장될 때 각 랙(rack)에 존재하는 SWM의 RTM은 자신이 관리하는 모든 IOP의 라우팅 테이블 정보를 관리하게 되며 각 IOP간의 포워딩 정보 어그리게이션(aggregation)을 랙(rack)간의 포워딩 정보 어그리게이션(aggregation)으로 쉽게 확장 할 수 있다.

<51> 도 5는 새롭게 삽입된 라우팅정보를 본 발명의 일 실시 예에 따라 관리하는 방법에 대한 처리 흐름도이다.

<52> 도 5를 참조하면, 도 3에 예시된 바와 같은 본 발명의 분산구조라우터에서 임의의 한 라우팅노드(RN)에 새로운 라우팅정보가 삽입되면(S110), 해당 라우팅노드는 그 삽입된 라우팅정보에 대응되는 노드(삽입노드(insertion node))의 위치를 탐색한다(S120). 즉, 삽입하려는 라우팅정보의 프리픽스(prefix) 길이와 같은 노드를 탐색할 때까지 기

설정된 어그리게이션 트리(aggregation tree)의 최상위노드(root node)로부터 이진 탐색(binary search)을 반복수행한다. 상기 과정(S120) 수행 결과 삽입노드(insertion node)의 위치를 찾았으면, 그 삽입노드(insertion node)의 위치에 가상노드(delegation node)가 존재하는지를 확인한다(S130). 그리고 상기 확인(S130) 결과 삽입노드(insertion node)의 위치에 가상노드(delegation node)가 존재하지 않으면, 즉 해당 위치가 비어있으면 그 삽입노드(insertion node)를 삽입한다(s200).

<53> 만일 상기 확인(S130) 결과 삽입노드(insertion node)의 위치에 가상노드(delegation node)가 존재하면 그 가상노드(delegation node)를 삭제하여 삽입노드(insertion node)의 위치를 확보한 후 삽입노드(insertion node)를 삽입하여야 한다.

<54> 이를 위해, 도 5의 예에서는 가상노드(delegation node)의 좌/우 서브트리(L/R subtree)가 존재하는지를 확인하여(S140) 가상노드(delegation node)의 좌/우 서브트리(L/R subtree)가 존재하지 않으면 포워딩테이블에서 가상노드(delegation node)에 대응되는 포워딩정보를 삭제한다(S170). 한편, 그렇지 않은 경우, 즉 가상노드(delegation node)의 좌/우 서브트리(L/R subtree)가 존재하는 경우, 상기 가상노드(delegation node)는 대응되는 좌/우 서브트리(L/R subtree)를 구성하는 노드들을 대표하는 노드이므로 그 가상노드(delegation node)를 삭제하기 전에 탐색된 좌/우 서브트리를 구성하는 노드들을 재삽입하는 과정(S150)을 수행하여야 한다. 이 때, 탐색된 좌/우 서브트리를 구성하는 노드들을 재삽입하기 위해서는 본 발명의 삽입과정전체(도 5)를 다시 수행한다. 이를 통상적으로 순환함수(recursive function)이라 한다. 즉, 상기 과정(S140)의 확인결과 가상노드의 좌/우 서브트리가 있다고 판단된 경우는 상기 재삽입과정(S150)을 수행한 후 포워딩테이블에서 가상노드에 대응되는 포워딩정보를 삭제한다(S160).

<55> 이와 같이 삽입노드(insetrion node)의 위치에 가상노드(delegation node)가 존재하는 경우는 상기 일련의 과정(S140 내지 S170)을 통해 그 가상노드(delegation node)를 삭제한 후 삽입노드(insertion node)를 삽입한다(S200).

<56> 라우팅 테이블에 새로운 정보가 삽입된 경우 그 대응되는 삽입노드를 삽입하는 구체적인 처리 과정(S200)은 도 6a 내지 도 6c에 예시되어 있다.

<57> 도 6a를 참조하면 삽입노드를 삽입하기에 앞서 해당 라우팅 노드는 어그리게이션 가능한 최대 레벨(defaultlevel)범위 내에서 삽입노드(insertion node)의 조상노드(ancestor node)를 찾는다(S201). 이는 어그리게이션 트리(aggregation tree)에 삽입노드(insertion node)를 대표하는 가상노드(delegation node)를 삽입하기 위한 위치정보를 얻기 위함이다.

<58> 상기 과정(S201) 결과 삽입노드(insertion node)의 조상노드(ancestor node)가 없으면, 삽입노드(insertion node)의 상위레벨-탐색범위(searchlevel)을 설정한다(S205). 상위레벨-탐색범위(searchlevel)는 (수학식 1)에 의해 산출하는 것이 바람직하다.

<59> 【수학식 1】 searchlevel = defaultlevel*2-1

<60> 그리고 그 상위레벨-탐색범위(searchlevel) 내에 삽입노드(insertion node)의 조상노드(ancestor node)가 있는지의 여부를 확인한다(S207). 이는 삽입노드(insertion node)를 대표하는 가상노드(delegation node) 삽입시의 오류를 줄이기 위함이다.

<61> 상기 확인(S207) 결과 해당 범위(searchlevel) 내에 삽입노드(insertion node)의 조상노드(ancestor node)가 없으면 초기에 설정된 어그리게이션 가능 레벨

(defaultlevel)에 삽입노드(insertion node)를 대표하는 가상노드(delegation node)를 삽입한다(S217). 만일 해당 범위(searchlevel) 내에 삽입노드(insertion node)의 조상노드(ancestor node)가 있으면 어그리게이션 가능한 최대레벨 범위(defaultlevel) 내에서 가상노드(delegation node)의 후손노드(descendant node)를 탐색한다(S209). 이는 가상노드(delegation node)에 의해 대표되는 후손노드(descendant node)가 있는지의 여부를 확인하기 위함이다.

<62> 상기 과정(S209) 수행결과 만일 상기 범위(defaultlevel) 내에 삽입노드(insertion node)의 후손노드(descendant node)가 없다면 과정(S217)을 수행한다. 즉, 초기에 설정된 어그리게이션 가능 레벨(defaultlevel)에 삽입노드(insertion node)의 가상노드(delegation node)를 삽입한다. 그리고 상기 과정(S209) 수행결과 상기 범위(defaultlevel) 내에 삽입노드(insertion node)의 후손노드(descendat node)가 있다면(S211) 어그리게이션 가능한 레벨을 재설정한(S213) 후 재설정된 어그리게이션 가능 레벨(dynamiclevel)에 가상노드(delegation node)를 삽입한다(S215). 이 때, 어그리게이션 가능한 레벨을 재설정하는 방법은 (수학식 2)에 예시된 방법을 이용하는 것이 바람직하다.

<63> 【수학식 2】 dynamiclevel = *GetNodePrefixLength*(insertion node)

<64> - *GetDistPrefixLength*(insertionnode, desendantnode)

<65> 이 때, '*GetNodePrefixLength*(insertion node)'는 삽입노드(insertion node)의 프리픽스(prefix) 길이를 얻는 함수이고, '*GetDistPrefixLength*(insertionnode, desendantnode)'는 삽입노드(insertion node)와 그 후손노드(desendant node)의 프리픽스의 차이가 나타나는 위치에 대한 정보를 얻는 함수이다. 예를 들어, 삽입노드(insertion node)의 프리픽스 길이가 '18'이고, 삽입노드(insertion node)와 그 후손노드(desendant node)의 프리픽스의 차이가 나타나는 위치가 '19'번째부터 라면, 상기 (수학식 2)에 의해 재설정된 어그리게이션 가능 레벨은 그 차이값인 '1'이 될 것이다. 따라서 삽입노드(insertion node)의 가상노드(delegation node)는 삽입노드(insertion node)의 레벨보다 하나 높은 상위 레벨에 생성하는 것이 가능한 것이다.

<66> 그리고, 상기 일련의 과정들(S205 내지 S217)에 의해 어그리게이션 트리의 소정 위치에 삽입노드(insertion node)를 대표하는 가상노드(delegation node)를 삽입하였으면 그 삽입된 라우팅정보를 포워딩테이블에 저장할 지의 여부를 판단하기 위해 라우팅정보의 생성영역을 확인한다(S219). 확인결과 라우팅정보의 생성영역이 가상영역이면 포워딩테이블에 라우팅정보를 대표하는 가상노드(delegation node)정보를 삽입하고(S221), 라우팅정보의 생성영역이 로컬영역이면 포워딩테이블에 라우팅정보에 대응되는 삽입노드(insertion node) 정보를 삽입한다(S223).

<67> 한편, 도 6b는 상기 과정(S201) 결과 삽입노드(insertion node)의 조상노드(ancestor node)가 있는 경우에 대한 처리 과정을 예시하고 있다.

<68> 도 6b를 참조하면, 삽입노드(insertion node)의 조상노드(ancestor node)가 이미 있으면, 소정의 알고리즘(SelectDelegationLevel)(도 6c에 도시됨)에 의해 가상노드 삽입레벨을 선택한다(S230). 이 때, 상기 알고리즘(SelectDelegationLevel)에서는 상기 삽

입노드(insertion node)의 조상노드(ancestor node)가 이미 포워딩테이블에 업데이트되었고, 삽입노드(insertion nod)와 그 조상노드(ancestor node)가 동일한 IOP에 의해 생성된 경우 포워딩테이블에 업데이트 하는 과정을 생략하기 위해 선택레벨 값을 가상의 값(-1)으로 설정한다(도 6c의 S235 내지 S237).

<69> 이와 같이 소정의 알고리즘(SelectDelegationLevel)에 의해 가상노드를 삽입하기 위한 레벨을 선택하였으면, 그 선택된 레벨(dynamiclevel)을 '0'과 비교한다(S241).

<70> 만일 그 레벨(dynamiclevel)이 '0'이라면 삽입노드(insertion node)를 대표할 가상노드(delegation node)를 삽입할 수 없는 경우이므로 포워딩테이블에 삽입노드(insertion node) 정보를 직접 삽입한다(S251).

<71> 그리고 그 레벨(dynamiclevel)이 '0' 보다 작다면 삽입된 라우팅정보의 생성영역을 다시 한번 확인하여(S253) 삽입된 라우팅정보의 생성영역이 로컬영역인 경우에만 포워딩테이블에 삽입노드(insertion node)정보를 업데이트한다(S255). 만일 삽입된 라우팅정보의 생성영역이 가상영역인 경우에는 포워딩테이블에 업데이트하는 과정없이 다음 단계로 진행한다. 이 경우가 어그리게이션에 의한 효과를 보게 되는 경우이다.

<72> 한편 그 레벨(dynamiclevel)이 '0' 보다 크다면 상기 과정(S230)에서 선택된 레벨에 삽입노드(insertion node)의 가상노드(delegation node)를 삽입한다(S243). 그리고 삽입된 라우팅정보의 생성영역을 확인하여(S245) 라우팅정보의 생성영역이 가상영역이면 포워딩테이블에 가상노드(delegation node) 정보를 삽입하고(S247), 라우팅정보의 생성영역이 로컬영역이면 포워딩테이블에 삽입노드(insertion node) 정보를 삽입한다(S249).

<73> 도 6c는 상기 과정(S230)에 대한 처리 흐름도이다. 도 6c를 참조하면 가상노드(delegation node)의 삽입레벨을 선택하기 위해, 본 발명에서는 삽입노드(insertion node)의 후손노드(descendant node)가 존재하는 지의 여부에 따라 어그리게이션 가능한 레벨을 서로 다르게 재설정한다. 즉, 우선 삽입노드(insertion node)의 후손노드(descendant node)를 탐색하여(S231) 삽입노드(insertio node)의 후손노드(descendant node)가 존재하면 상기 (수학식 2)에 예시된 방법에 의해 어그리게이션 가능한 레벨을 재설정한다(S233). 즉, 삽입노드(insertion node)의 프리픽스 길이 및 삽입노드(insertion node)와 그 후손노드(desendant node)의 프리픽스의 차이가 나타나는 위치 정보에 의거하여 어그리게이션 가능한 레벨을 재설정한다. 한편, 상기 탐색결과(S231)를 확인(S232)하여 삽입노드(insertio node)의 후손노드(descendant node)가 존재하지 않으면 삽입노드(insertion node)의 조상노드(ansector node)의 레벨을 어그리게이션 가능 레벨로 설정한다(S234).

<74> 한편, 상기 삽입노드(insertion node)의 조상노드(ancestor node)가 이미 포워딩테이블에 업데이트되었고(S235), 삽입노드(insertion nod)와 그 조상노드(ancestor node)가 동일한 IOP에 의해 생성된 경우(S236) 도 6b에서 포워딩테이블 업데이트 과정을 생략하기 위해 선택레벨 값을 가상의 값(-1)으로 설정한다(S237).

<75> 도 7은 삭제된 라우팅정보를 본 발명의 일 실시 예에 따라 관리하는 방법에 대한 처리 흐름도이다.

<76> 도 7을 참조하면, 라우팅 노드들을 다수개 포함하는 분산구조라우터에서 임의의 한 라우팅노드(RN)의 라우팅테이블에 기 저장된 라우팅정보가 삭제되면(S510), 그 라우팅노드(RN)는 삭제된 라우팅정보에 대응되는 노드(삭제노드(deletion node))의 위치를 탐색

한다(S50). 이 과정은 라우팅테이블에 새로운 라우팅정보가 삽입된 경우 그 삽입노드의 위치를 탐색하는 과정과 유사하다. 즉, 삭제하려는 라우팅정보의 프리픽스(prefix) 길이와 같은 노드를 탐색할 때까지 기 설정된 어그리게이션 트리(aggregation tree)의 최상위노드(root node)로부터 이진 탐색(binary search)을 반복수행한다.

<77> 상기 과정(S520) 수행 결과 삭제노드(deletion node)의 위치를 찾았으면, 그 삭제노드(deletion node)가 가상노드(delegation node)를 생성한 노드인지의 여부를 확인하여(S530), 그 삭제노드(deletion node)가 가상노드(delegation node)를 생성한 노드이면 그 삭제노드가 생성한 가상노드(delegation node) 정보를 삭제노드(deletion node)로 변경한다(S540). 이는 삭제노드(deletion node)가 임의의 가상노드(delegation node)에 의해 대표되고있는 경우 그 가상노드(delegation node)를 삭제함으로써 실질적으로 삭제노드(deletion node)의 삭제와 동일한 효과를 가져오도록 하기 위함이다.

<78> 그리고, 삭제노드(deletion node)가 포워딩테이블에 업데이트(update)되었는지의 여부를 확인하여(S550) 그 삭제노드(deletion node)가 포워딩 테이블에 업데이트(update)되지 않은 경우 더 이상 작업을 수행하지 않고 삭제 과정을 종료한다. 한편, 그 삭제노드(deletion node)가 포워딩 테이블에 업데이트(upeate)되었다면 어그리게이션 가능한 최대 레벨 범위 내에서 삭제노드(deletion node)의 후손노드(descendant node)를 탐색하여(S560), 그 범위 내에 삭제노드(deletion node)의 후손노드(descendant node)가 있으면 후손노드(descendant node)를 가상노드(delegation node)의 새로운 소유자로 변경한다(S580). 이는 하나의 가상노드(delegation node)가 다수의 후손노드(descendant node)를 대표하고 있는 경우 최초에 가상노드(delegation node)를 생성한 후손노드

(descendant node)의 삭제에 의해 나머지 후손노드(descendant node)들을 대표하는 가상 노드(delegation node)의 부재를 막기 위함이다.

<79> 만일 상기 확인(S570) 결과 그 범위 내에 삭제노드(deletion node)의 후손노드(descendant node)가 없으면 포워딩테이블에서 삭제노드(deletion node)를 삭제한다(S590).

<80> 도 8a 내지 도 8g는 본 발명의 일 실시 예에 따른 포워딩정보를 동적으로 관리하는 방법에 대한 알고리즘 예를 나타낸 도면이다.

<81> 도 8a는 도 5의 처리 과정들을 의사코드(pseudo code)로 표현하였고, 도 8b는 도 5의 과정 S120 및 도 7의 과정 S520을 의사코드(pseudo code)로 표현하였고, 도 8c는 도 6a의 처리과정들을 의사코드(pseudo code)로 표현하였고, 도 8d는 도 6a의 과정 S215 및 S217에서 가상 노드를 삽입하는 과정을 의사코드(pseudo code)로 표현하였고, 도 8e는 도 6b의 처리과정들을 의사코드(pseudo code)로 표현하였고, 도 8f는 도6c의 처리과정들을 의사코드(pseudo code)로 표현하였고, 도 8g는 도 7의 처리과정들을 의사코드(pseudo code)로 표현하였다. 각 처리 과정들에 대한 구체적인 설명은 도 5 내지 도 7를 참조한 설명부분에 언급되었으므로 그 구체적인 설명은 생략한다. 도 8a 내지 도 8g에 예시된 알고리즘 처리 방식은 하나의 실시 예에 불과한 것으로서, 이러한 알고리즘을 다양하게 변형하여 실행할 수 있음은 자명하다.

<82> 도 9a 내지 도 10b는 본 발명에 의한 포워딩정보 관리 방법을 도식화하여 설명한 도면으로서 디폴트 레벨(defaultlevel)이 '2'인 경우에 새롭게 삽입된 포워딩

정보 및 삭제된 포워딩정보를 관리하는 방법을 설명하고 있다. 도 9a 및 도 9b는 새롭게 삽입된 포워딩정보를 관리하는 방법을 도식화하여 설명한 도면이고, 도 10a 및 도 10b는 삭제된 포워딩정보를 관리하는 방법을 도식화하여 설명한 도면이다.

<83> 도 9a 내지 도 10b의 어그리게이션 트리에서 'Prefix'는 해당 노드에 대응되는 포워딩정보가 포워딩될 주소정보를 나타내는 것으로서 노드 식별기능을 가진다. 'Source IOP'는 해당 노드를 생성한 IOP 정보를 나타내고, 'FE'는 해당 포워딩정보를 포워딩테이블에 저장하였는지의 여부를 나타낸다. 따라서 'FE'의 빗금은 해당 포워딩정보를 포워딩테이블에 저장하였음을 나타낸다. 또한 'Index'는 그 노드의 소유자(예컨대, 그 노드가 가상노드인 경우 가상노드를 생성한 노드)정보를 나타내고, 'level'은 해당 노드를 어그리게이션할 수 있는 레벨(aggregation-level)을 나타낸다.

<84> 우선 도9a를 참조하면, 아무 정보도 등록되지 않은 상태에서 IOP#1(210)에 포워딩 주소(prefix)가 '3'인 새로운 라우팅정보가 삽입된 경우 IPO#1(210)의 제어부는 그 정보를 IOP#1(210)의 어그리게이션 트리에 삽입한다. 즉, 노드3을 어그리게이션 트리에 삽입한다. 그리고, 그 노드에 대한 어그리게이션 가능 레벨이 '2' 이므로 2레벨 상위에 포워딩주소(prefix)가 '0'인 가상노드(delegation node)를 생성하여 어그리게이션 트리에 추가한다. 즉, 노드0(가상노드)을 어그리게이션 트리에 삽입한다. 이 때, 포워딩주소(prefix)가 '0'인 가상노드(delegation node)의 소유자는 포워딩주소(prefix)가 '3'인 노드가 되므로 포워딩주소(prefix)가 '0'인 가상노드(delegation node)의 소유자정보(index)는 '00'이 된다. 그리고, 이들의 소스 IOP가 모두 IOP#1(210)이므로 IOP#1(210)의 포워딩테이블에는 실제 삽입노드 정보인 포워딩주소(prefix)가 '3'인 노드 정보가 저장된다.

<85> 한편, 본 발명의 분산구조라우터는 모든 노드들의 라우팅정보를 각 노드들이 모두 공유하는 특성이 있으므로, IOP#1(210)이외의 다른 노드들은 IOP#1(210)에 저장된 라우팅정보(예컨대, 어그리게이션 트리)와 동일한 라우팅 정보(예컨대, 어그리게이션 트리)를 가지게 된다. 도 9a의 예에서는 IOP#1(210)이외의 다른 노드에 대한 예로 IOP#2(220)를 표시하였다. 이는 이후의 예에서도 동일하게 적용된다.

<86> 따라서, 도 9a를 참조하면, IOP#2(220)의 어그리게이션 트리는 IOP#1(210)의 어그리게이션 트리와 동일하고, 다만 포워딩테이블 정보만이 다르다. 즉, IOP#1(210)은 로컬영역에서 새로운 라우팅정보가 삽입되었으므로 삽입노드('3')정보를 포워딩테이블에 저장하고, IOP#2(220)는 가상영역에서 새로운 라우팅정보가 삽입되었으므로 가상노드('0')정보를 포워딩테이블에 저장하였다.

<87> 도 9b는 IOP#1(210)에 포워딩주소(prefix)가 '3'인 라우팅정보가 삽입되어 그를 대표하는 가상노드(delegation node)(노드0)가 IOP#2(220)의 포워딩테이블에 저장된 상태에서, IOP#2(220)의 로컬영역에 포워딩주소(prefix)가 '7'인 라우팅정보가 삽입된 경우에 대한 예를 나타내고 있다.

<88> 이 경우 IOP#2(220)의 제어부는 삽입노드(insertion node)(7)의 위치를 탐색한 후, 디폴트 레벨(default level)값(2)에 의해 삽입노드(insertion node)의 조상노드(ancestor node)를 탐색한다. 그러면 IOP#2(220)의 제어부는 노드1을 삽입노드(노드7)의 조상노드로 찾게 된다. 이 때, 노드1은 IOP#1(210)의 로컬영역에 삽입된 노드3을 찾기 위해 거치게 되는 노드이다. 따라서, 노드7에 대한 가상노드는 노드1이 아닌 노드 4의 위치에 삽입된다. 본 발명에서는 이와 같이 어그리게이션 가능한 최대 레벨이 가변적이

며 이와 같이 가변적으로 재설정된 어그리게이션 가능 레벨을 다이나믹레벨(dynamiclevel)이라고 명명한다.

<89> 그러면, 도 9b에 예시된 바와 같이 IOP#1(210)의 어그리게이션 트리에는 노드7과 노드4가 모두 추가되며, 노드4 및 노드7의 생성영역이 IOP#1(210) 로컬영역이 아닌 가상의 영역이므로 노드7의 가상노드인 노드 4를 IOP#1(210)의 포워딩테이블에 업데이트한다.

<90> 한편, IOP#2(220)의 제어부는 그 로컬영역에 노드7이 삽입된 경우이므로 노드7에 대한 정보를 포워딩테이블에 업데이트한다.

<91> 도 10a 및 도 10b를 참조하여 본 발명에 의한 라우팅정보 삭제 과정을 예시적으로 설명하면 다음과 같다.

<92> 먼저 도10a에 예시된 바와 같이 어그리게이션 트리의 노드0은 IOP#1(210)의 로컬영역에서 생성된 노드3, 노드4, 노드5 및 노드6을 모두 대표하고 있으며 노드0의 소유자는 노드3인 경우, IOP#1(210)의 포워딩테이블에는 노드3, 노드4, 노드5 및 노드 6에 대한 정보가 업데이트되어 있고 IOP#2(220)의 포워딩테이블에는 그 노드들(노드3, 노드4, 노드5 및 노드6)을 모두 대표하는 노드0만이 업데이트되어 있다.

<93> 도 10b는 도 10a의 예에서 노드0의 소유자인 노드3이 삭제된 경우를 도식화하여 설명하고 있다.

<94> 본 발명에서는 이와 같이 가상노드의 소유자가 되는 노드가 삭제되는 경우 로컬영역에서는 그 노드 정보를 삭제한 후 그 노드 정보를 대표하는 가상노드의 소유자 정보만을 변경하고, 가상영역에서는 아무 작업도 수행하지 않는다.

<95> 즉, 도 10b에 예시된 바와 같이 노드0의 소유자인 노드3이 삭제된 경우, IOP#1(210)의 제어부는 노드0의 소유자를 노드4로 변경해준다. 한편, IOP#2(220)의 제어부는 IOP#2(220)의 포워딩테이블에 대하여 별도의 작업을 수행하지 않아도 된다.

<96> 도 11a 내지 도 11d는 본 발명의 일 실시 예에 따른 포워딩정보 동적관리 방법의 효과를 입증하기 위한 테스트 결과들이다.

<97> 본 발명에서는 이러한 테스트 결과를 얻기 위해 1개의 SWM과 2개의 IOP를 갖춘 Galaxy 시스템과 "http://www.telstra.net/ops/bgptable.html"에서 제공되는 core BGP 라우터의 라우팅 테이블 엔트리를 이용하여 53,000개의 라우팅 엔트리를 삽입하는 경우와 이중의 일정 비율을 플랩(flap)하는 테스트를 수행하였다.

<98> 도 11a는 본 발명의 효과가 삽입되는 엔트리의 순서에 영향을 받는지를 측정하기 위한 테스트 결과를 도시한 도면으로서, 10번에 걸쳐서 오름차순으로 정렬된 샘플 엔트리의 순서를 무작위로 변경하면서 6개의 defaultlevel 값을 적용하여 그 샘플 엔트리를 삽입해본 결과를 나타낸다. 도 11a를 참조하면, 상기 테스트 결과 포워딩엔트리(forwarding entries)의 개수는 60개 ~ 110개 이내이다. 따라서, 본 발명의 성능은 엔트리들의 삽입순서에는 거의 영향을 받지 않는 다는 것을 알 수 있다.

<99> 도 11b는 6가지의 defaultlevel을 적용하였을 때의 오버헤드 측정결과를 도시한 도면이다. 도 11a 및 도 11b를 참조하면 라우터의 성능과 오버헤드를 고려할 때 본 발명은 defaultlevel이 '3'이하일 때 가장 효과적이라는 것을 알 수 있다.

<100> 도 11c는 53,000개의 core BGP 라우팅 엔트리를 각각 0 ~ 70% 삭제하면서 포워딩 테이블에 존재하는 포워딩 엔트리의 개수를 측정한 결과를 도시한 도면이다. 도 11c를

참조하면 본 발명은 core BGP 엔트리를 단순 삽입하고 삭제비율이 0%인 경우 약 26% - 77%의 포워딩 테이블 엔트리 감소 효과가 있다는 것을 알 수 있다.

<101> 도 11d는 본 발명이 실제 인터넷 환경에서 어떠한 성능을 나타내는지 측정하기 위해서 Agilent Router Tester를 이용하여 실제 인터넷 환경의 성능을 측정한 결과를 도시하고 있다. 일반적으로 Router Tester는 네트워크 토폴로지(topology)나 네트워크가 영향을 미치는 범위(reachability)가 지속적으로 변하는 동적인 라우팅조건에서 라우터의 성능을 측정할 수 있다. 도 11d는 라우터가 라우팅 정보의 변화를 포워딩 테이블에 반영하는 시간(convergence time)에 초점을 맞추어 테스트 결과를 도시하고 있다. 이러한 " BGP4 flap convergence time" 테스트는 네트워크 토폴로지(topology)나 네트워크가 영향을 미치는 범위(reachability)의 변화에 의해 주 라우터(primary router)가 드롭(drop)된 경우 새로운 라우터로 변경하는 시간(traffic redirect convergence time)을 측정한다. 이 측정에서는 3개의 라우팅노드(RN)를 사용하는데, 2개의 라우팅노드(RN)들이 그들 뒤에 있는 동일한 네트워크에 대한 영향력범위(reachability)를 통지(advertise)하면 나머지 1개의 라우팅노드(RN)는 통지(advertised)된 네트워크에 데이터 트래픽(data traffic)을 생성한다. 도 11d는 이러한 테스트의 수행시간(convergence time)결과를 보여 준다. 도 11d의 예에서 본 발명에 의해 상기 수행시간(convergence time)이 줄어든 효과를 보이는 것은 포워딩 테이블 수정의 최소화를 위해 가상노드의 실제 소유자 삭제시 그 가상노드의 소유자 정보만을 변경하는 경우이다.

<102> 상술한 본 발명의 설명에서는 구체적인 실시 예에 관해 설명하였으나, 여러 가지 변형이 본 발명의 범위에서 벗어나지 않고 실시할 수 있다. 따라서 본 발명의 권리범위

는 설명된 실시 예에 의하여 한정되는 것이 아니고 특허청구범위와 특허청구범위의 균등한 것에 의해 정해 져야 한다.

【발명의 효과】

<103> 상술한 바와 같이 본 발명은 분산구조라우터의 모든 노드들에서 수집되는 라우팅정보들을 각 노드들이 실시간으로 공유하고 그 라우팅정보들을 이용하여 포워딩정보를 동적으로 관리함으로써, 라우팅정보를 공유하기 위한 각 노드들간의 패킷 포워딩 과정을 생략할 수 있다는 효과가 있다. 그리고 그 노드 정보들을 포워딩테이블에 선택적으로 업데이트 합으로써 노드별 포워딩테이블을 효율적으로 관리할 수 있다는 장점이 있다. 또한 본 발명은 2진 어그리게이션 트리(aggregation tree)를 이용하여 각 노드들의 포워딩정보를 관리하고, 라우팅정보에 해당되는 노드정보들을 어그리게이션하는 가상노드의 레벨을 가변적으로 설정함으로써 분산구조라우터의 노드별 포워딩테이블의 크기를 줄일 수 있다는 장점이 있다.

【특허청구범위】

【청구항 1】

라우팅노드들을 다수 포함하는 분산구조라우터에 있어서,

상기 라우팅노드들에 각각 포함된 다수의 라우팅 프로토콜에 대응되는 라우팅 프로토콜를 포함하여 상기 라우팅노드들에서 수집된 라우팅정보들을 실시간으로 공유하는 스위칭모듈을 포함하는 것을 특징으로 하는 분산구조라우터.

【청구항 2】

다수의 라우팅노드들에서 각각 수집된 라우팅정보들에 대한 라우팅테이블 및 그 라우팅테이블에 대한 어그리게이션 트리정보를 모든 라우팅노드들이 실시간으로 공유하는 분산구조라우터에서 포워딩정보를 관리하는 방법에 있어서,

라우팅테이블에 새로운 라우팅정보가 삽입되면 그 라우팅정보에 대응되는 삽입노드의 위치를 탐색하는 제1 과정과,

기 설정된 어그리게이션 가능한 최대레벨 범위 내에서 삽입노드의 조상노드를 탐색하는 제2 과정과,

상기 제2 과정의 탐색결과 상기 범위 내에 삽입노드의 조상노드가 존재하는 경우 그 조상노드가 포워딩테이블에 업데이트되었고 삽입노드와 조상노드의 생성영역이 동일하면 그 삽입노드 정보를 포워딩테이블에 업데이트하지 않는 제3 과정과,

상기 제2 과정의 탐색결과 상기 범위 내에 삽입노드의 조상노드가 존재하지 않는 경우 기 설정된 어그리게이션 가능한 최대레벨 범위 내에서 어그리게이션 가능레벨을 재

설정한 후 그 재설정된 어그리게이션 가능레벨에 삽입노드를 대표하는 가상노드를 삽입하는 제4 과정과,

상기 라우팅테이블에 삽입된 라우팅정보의 생성영역을 확인하여 그 생성영역이 가상영역이면 상기 가상노드 정보를 포워딩테이블에 삽입하고, 그 생성영역이 로컬영역이면 상기 삽입노드 정보를 포워딩테이블에 삽입하는 제5 과정을 포함하는 것을 특징으로 하는 분산구조라우터에서 포워딩정보를 관리하는 방법.

【청구항 3】

제2항에 있어서, 상기 제1 과정에서 탐색된 삽입노드의 위치에 가상노드가 존재하면, 그 가상노드에 대응되는 포워딩정보를 포워딩테이블에서 삭제하는 제6 과정을 더 포함하는 것을 특징으로 하는 분산구조라우터에서 포워딩정보를 관리하는 방법.

【청구항 4】

제3항에 있어서, 상기 제6 과정은 상기 가상노드의 좌/우 서브트리가 존재하는 경우 그 좌/우 서브트리 노드들을 재삽입하는 제6-1 과정을 더 포함하는 것을 특징으로 하는 분산구조라우터에서 포워딩정보를 관리하는 방법.

【청구항 5】

제2항에 있어서, 상기 제3 과정은

상기 제2 과정의 탐색결과 상기 범위 내에 삽입노드의 조상노드가 존재하는 경우 그 삽입노드의 후손노드를 탐색하는 제3-1 과정과,

상기 후손노드가 존재하면 그 후손노드와 삽입노드의 포워딩주소정보(prefix)값의 차이에 의해 어그리게이션 가능레벨을 재설정하고, 그렇지 않으면 삽입노드의 조상노드의 레벨값에 의해 어그리게이션 가능레벨을 재설정하는 제3-2 과정과,

상기 재설정된 어그리게이션 가능레벨이 '0'이면 포워딩테이블에 삽입노드정보를 삽입하는 제3-3 과정과,

상기 재설정된 어그리게이션 가능레벨이 '0'보다 크면 상기 재설정된 어그리게이션 가능레벨에 삽입노드를 대표하는 가상노드를 삽입하는 제3-4 과정과,

상기 삽입된 라우팅정보의 생성영역을 확인하여 상기 라우팅정보의 생성영역이 가상영역이면 포워딩테이블에 상기 제3-4 과정에서 삽입된 가상노드 정보를 삽입하고, 상기 라우팅정보의 생성영역이 로컬영역이면 포워딩테이블에 삽입노드정보를 삽입하는 제3-5 과정을 포함하는 것을 특징으로 하는 분산구조라우터에서 포워딩정보를 관리하는 방법.

【청구항 6】

제2항에 있어서, 상기 제4 과정은

삽입노드의 상위에 조상노드가 있는지의 여부를 판단하기 위한 상위레벨탐색범위를 설정하는 제4-1 과정과,

상기 상위레벨탐색범위 내에 삽입노드의 조상노드가 있으면 기 설정된 어그리게이션 가능 최대레벨 범위 내에서 삽입노드를 대표하게될 가상노드의 후손노드가 있는지의 여부를 판단하는 제4-2 과정과,

상기 제4-2 과정의 판단결과 상기 어그리게이션 가능 최대레벨 범위 내에서 가상노드의 후손노드가 있으면 그 후손노드와 삽입노드의 포워딩주소정보(prefix)값의 차이에 의해 어그리게이션 가능레벨을 재설정하는 제4-3 과정과,

상기 재설정된 어그리게이션 가능레벨에 삽입노드의 가상노드를 삽입하는 제4-4 과정을 포함하는 것을 특징으로 하는 분산구조라우터에서 포워딩정보를 관리하는 방법.

【청구항 7】

다수의 라우팅노드들에서 각각 수집된 라우팅정보들에 대한 라우팅테이블 및 그 라우팅테이블에 대한 어그리게이션 트리정보를 모든 라우팅노드들이 실시간으로 공유하는 분산구조라우터에서 포워딩정보를 관리하는 방법에 있어서,

라우팅테이블에 기 저장된 라우팅정보가 삭제되면 그 라우팅정보에 대응되는 삭제노드의 위치를 탐색하는 제1 과정과,

상기 삭제노드가 포워딩테이블에 업데이트된 경우 어그리게이션 가능한 최대레벨 범위내에서 상기 삭제노드의 후손노드를 탐색하는 제2 과정과,

상기 제2 과정의 탐색결과 상기 범위 내에 상기 삭제노드의 후손노드가 있으면 후손노드를 가상노드의 새로운 소유자로 변경하고, 그렇지 않은 경우 포워딩테이블에서 상

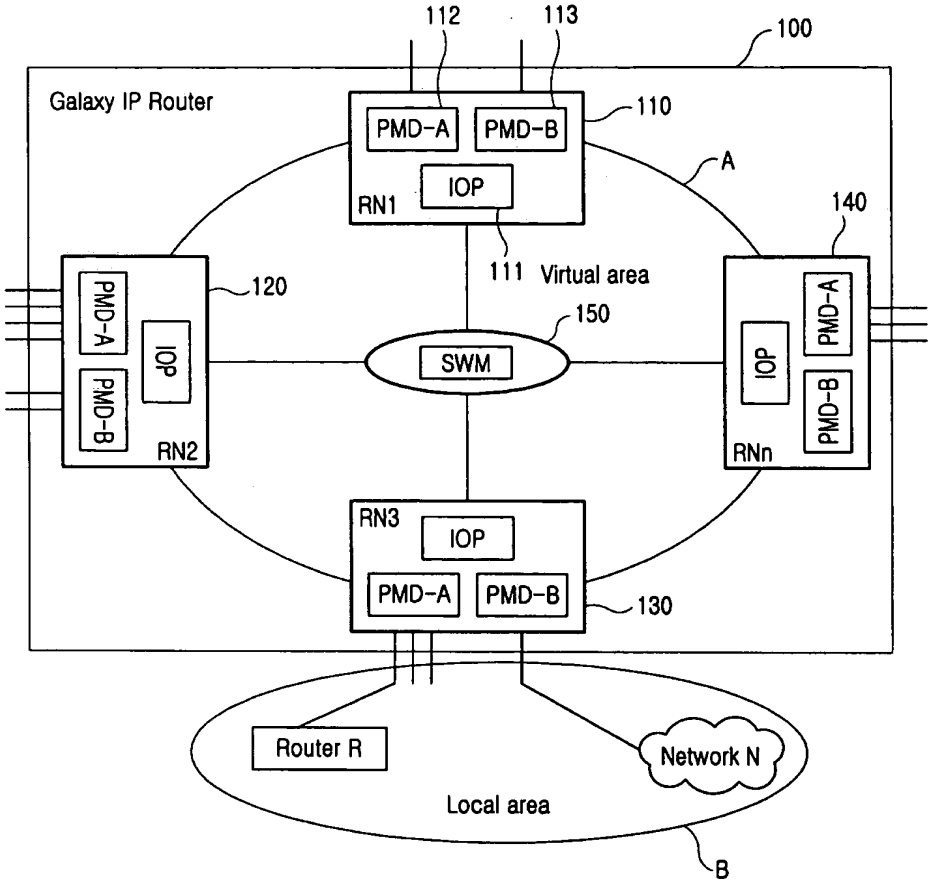
기 삭제노드를 삭제하는 제3 과정을 포함하는 것을 특징으로 하는 분산구조라우터에서 포워딩정보를 관리하는 방법.

【청구항 8】

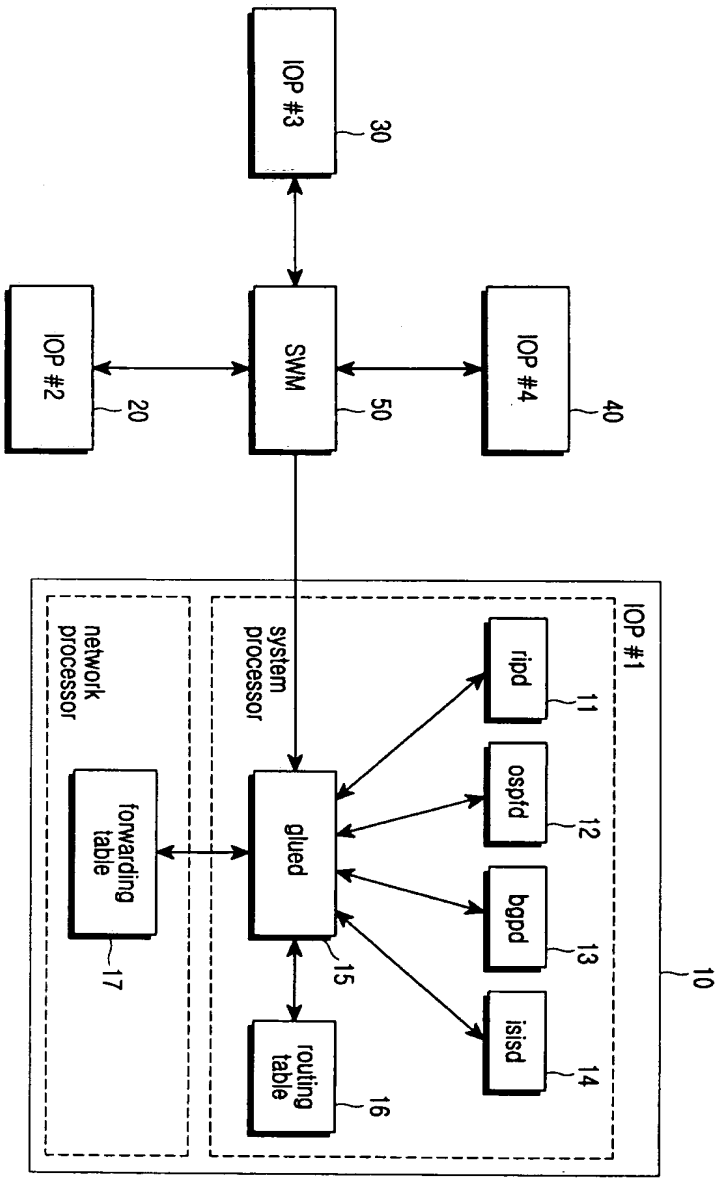
제7항에 있어서, 상기 삭제노드가 가상노드를 생성한 노드이면 상기 삭제노드가 생성한 가상노드 정보에 의해 삭제노드 정보를 변경하는 제4 과정을 더 포함하는 것을 특징으로 하는 분산구조라우터에서 포워딩정보를 관리하는 방법.

【도면】

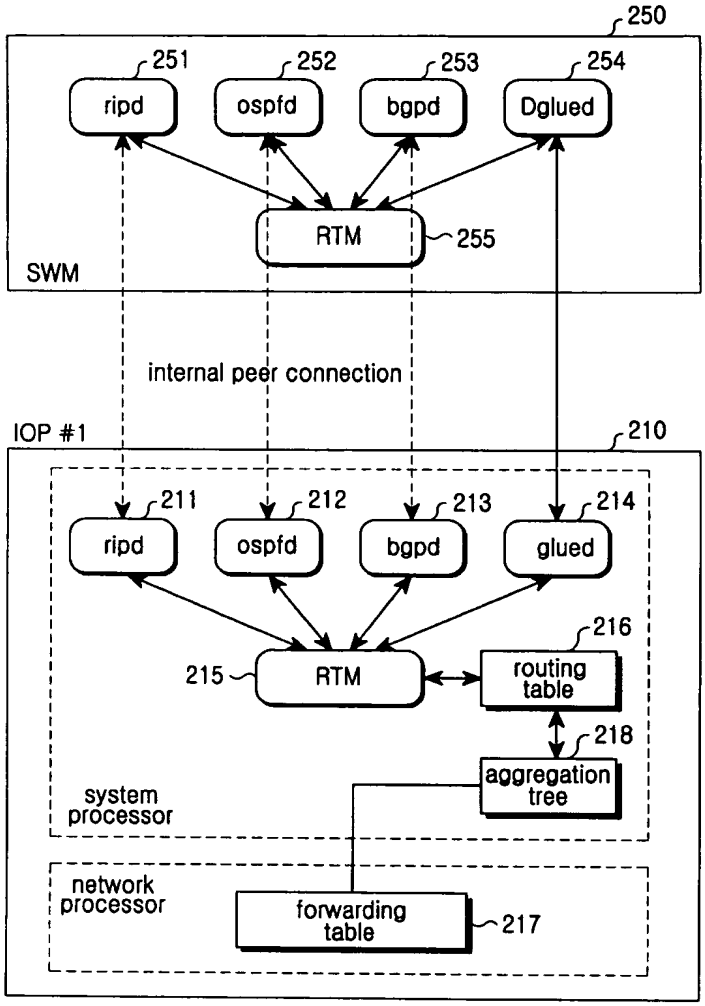
【도 1】


112
113
100
Galaxy IP Router
PMD-A
PMD-B
110
IOP
RN1
A
140
111 Virtual area
120
150
SWM
PMD-A
PMD-B
IOP
RN2
IOP
PMD-A
PMD-B
RNn
RN3
IOP
PMD-A
PMD-B
130
Router R
Network N
Local area
B

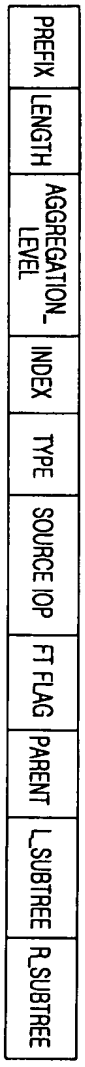
【도 2】


IOP #3
30
IOP #2
20
SWM
50
IOP #4
40
IOP #1
10
network processor
forwarding table
17
system processor
glued
15
routing table
16
ripd
11
ospfd
12
bgpd
13
isisd
14

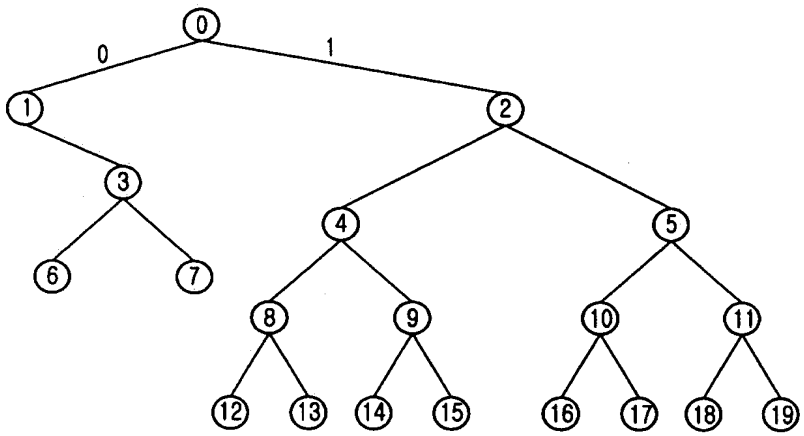
【도 3】


250
251
252
253
254
ripd
ospfd
bgpd
Dglued
RTM
255
SWM
internal peer connection
IOP #1
210
211
212
213
214
ripd
ospfd
bgpd
glued
216
215
RTM
routing table
218
aggregation tree
system processor
network processor
forwarding table
217

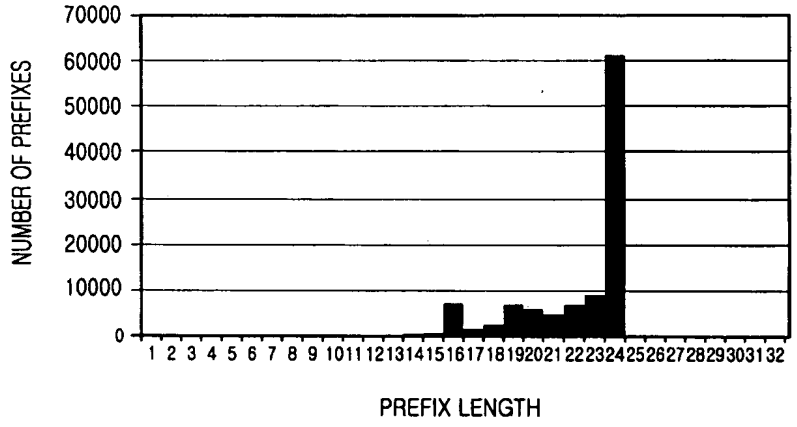
【도 4a】


PREFIX
LENGTH
AGGREGATION_LEVEL
INDEX
TYPE
SOURCE IOP
FT FLAG
PARENT
L_SUBTREE
R_SUBTREE

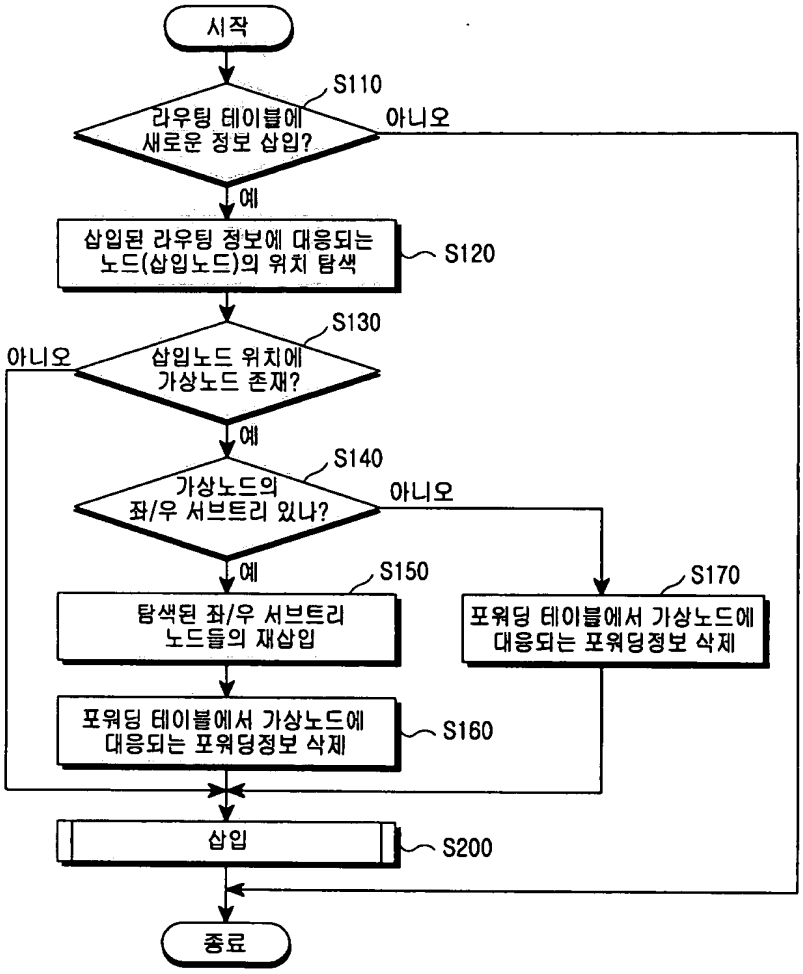
【도 4b】


0
0
1
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

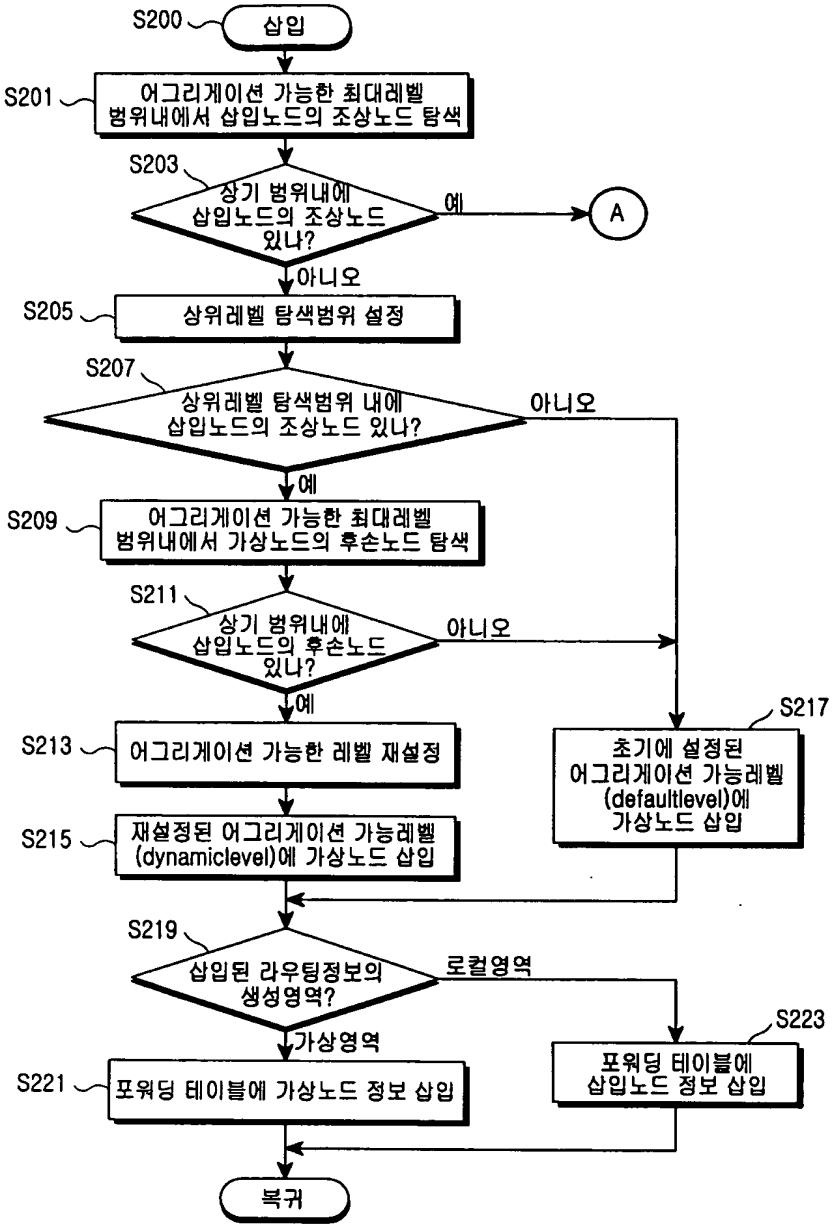
【도 4c】


NUMBER OF PREFIXES
70000
60000
50000
40000
30000
20000
10000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
PREFIX LENGTH

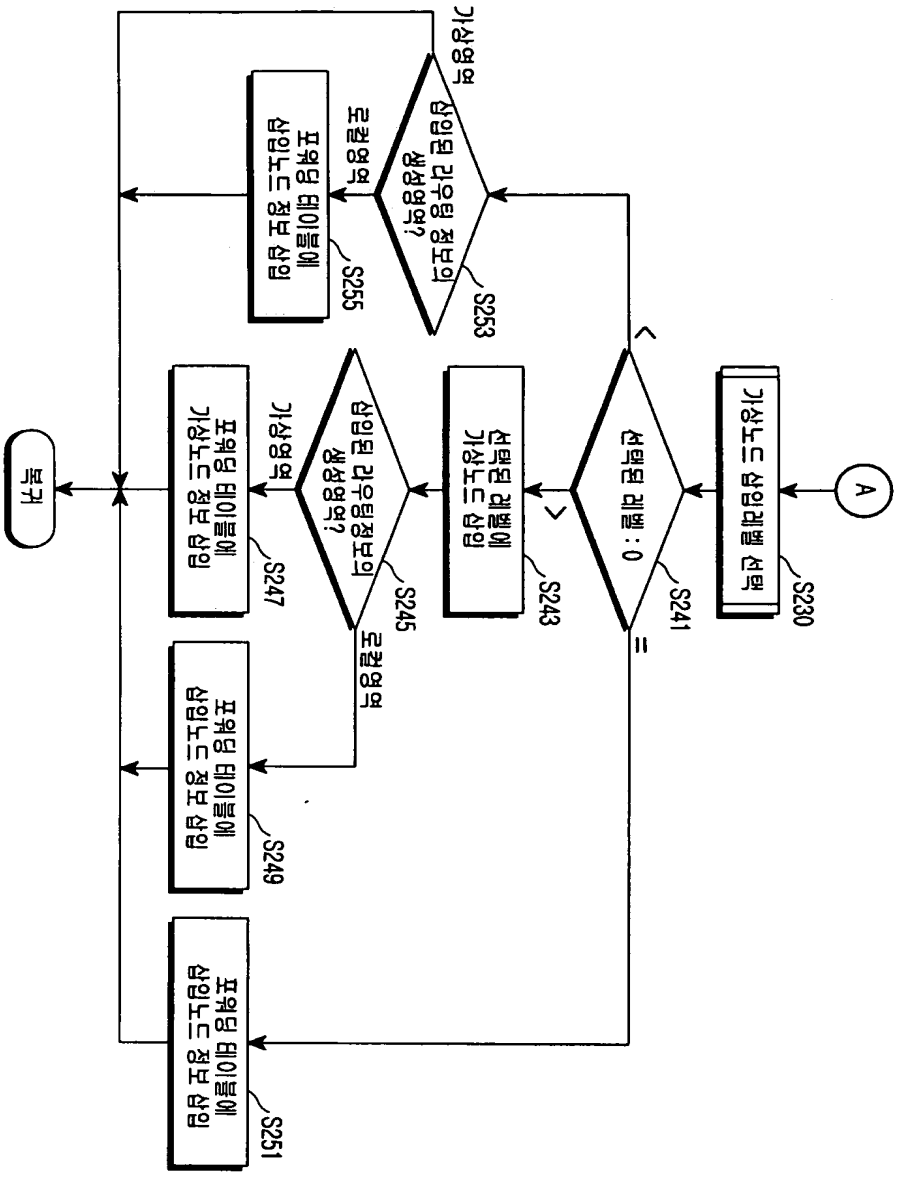
【도 5】


시작
S110
라우팅 테이블에 새로운 정보 삽입?
아니오
예
삽입된 라우팅 정보에 대응되는 노드(삽입노드)의 위치 탐색
S120
S130
삽입노드 위치에 가상노드 존재?
아니오
예
S140
가상노드의 좌/우 서브트리 있나?
아니오
예
S150
탐색된 좌/우 서브트리 노드들의 재삽입
S170
포워딩 테이블에서 가상노드에 대응되는 포워딩정보 삭제
포워딩 테이블에서 가상노드에 대응되는 포워딩정보 삭제
S160
삽입
S200
종료

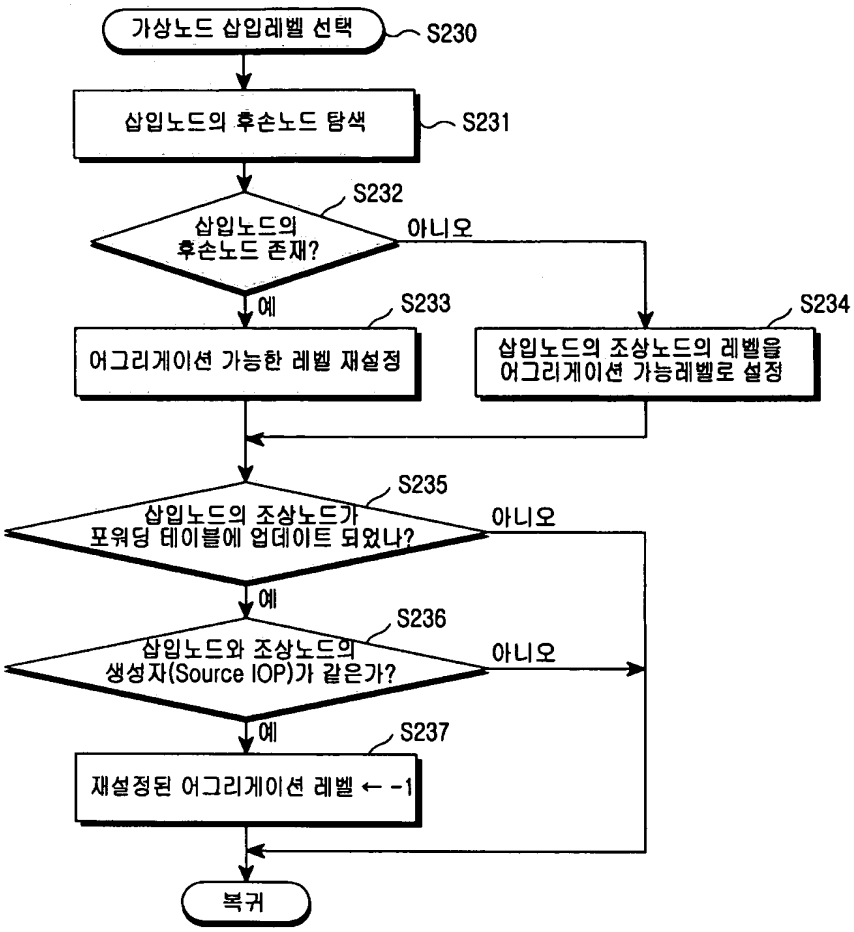
【도 6a】


S200
삽입
S201
어그리게이션 가능한 최대레벨 범위내에서 삽입노드의 조상노드 탐색
S203
상기 범위내에 삽입노드의 조상노드 있나?
예
A
아니오
S205
상위레벨 탐색범위 설정
S207
상위레벨 탐색범위 내에 삽입노드의 조상노드 있나?
아니오
예
S209
어그리게이션 가능한 최대레벨 범위내에서 가상노드의 후손노드 탐색
S211
상기 범위내에 삽입노드의 후손노드 있나?
아니오
예
S213
어그리게이션 가능한 레벨 재설정
S215
재설정된 어그리게이션 가능레벨 (dynamiclevel)에 가상노드 삽입
S217
초기에 설정된 어그리게이션 가능레벨 (defaultlevel)에 가상노드 삽입
S219
삽입된 라우팅정보의 생성영역?
로컬영역
가상영역
S221
포워딩 테이블에 가상노드 정보 삽입
S223
포워딩 테이블에 삽입노드 정보 삽입
복귀

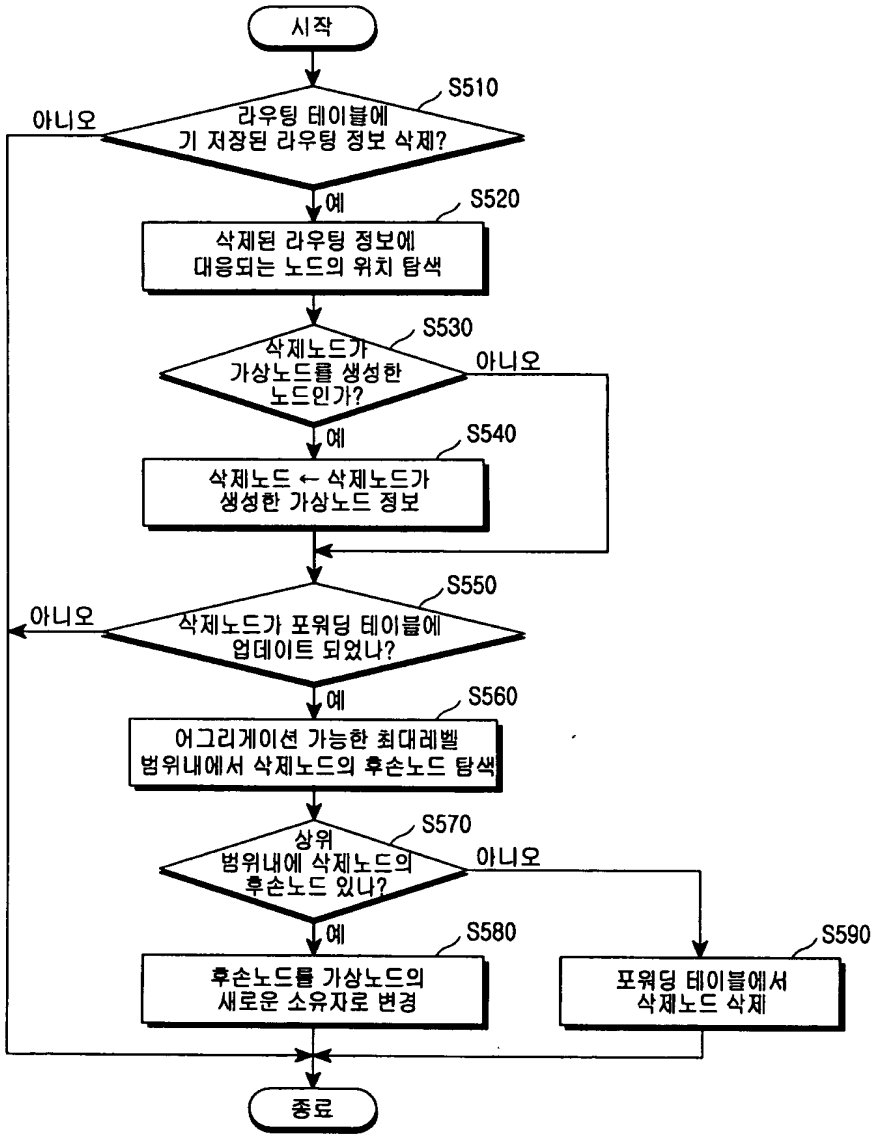
【도 6b】


가상노드 삽입레벨 선택
S230
선택된 레벨 : 0
S241
<
=
>
선택된 레벨에 가상노드 삽입
S243
삽입된 리우팅 정보의 생성영역?
S245
가상영역
로컬영역
표워딩 테이블에 가상노드 정보 삽입
S247
표워딩 테이블에 삽입노드 정보 삽입
S249
삽입된 리우팅 정보의 생성영역?
S253
가상영역
로컬영역
표워딩 테이블에 삽입노드 정보 삽입
S255
표워딩 테이블에 삽입노드 정보 삽입
S251
복귀
A

【도 6c】


가상노드 삽입레벨 선택
S230
삽입노드의 후손노드 탐색
S231
S232
삽입노드의 후손노드 존재?
아니오
예
S233
어그리게이션 가능한 레벨 재설정
S234
삽입노드의 조상노드의 레벨을 어그리게이션 가능레벨로 설정
S235
삽입노드의 조상노드가 포워딩 테이블에 업데이트 되었나?
아니오
예
S236
삽입노드와 조상노드의 생성자(Source IOP)가 같은가?
아니오
예
S237
재설정된 어그리게이션 레벨 ← -1
복귀

【도 7】


시작
S510
라우팅 테이블에 기 저장된 라우팅 정보 삭제?
아니오
예
S520
삭제된 라우팅 정보에 대응되는 노드의 위치 탐색
S530
삭제노드가 가상노드를 생성한 노드인가?
아니오
예
S540
삭제노드 ← 삭제노드가 생성한 가상노드 정보
S550
삭제노드가 포워딩 테이블에 업데이트 되었나?
아니오
예
S560
어그리게이션 가능한 최대레벨 범위내에서 삭제노드의 후손노드 탐색
S570
상위 범위내에 삭제노드의 후손노드 있나?
아니오
예
S580
후손노드를 가상노드의 새로운 소유자로 변경
S590
포워딩 테이블에서 삭제노드 삭제
종료

【도 8a】

```
Insertion (prefix) {
  local insertionnode, ancestornode, result
  /* STEP 1 */
  insertionnode := FindNode (prefix)
  /* STEP 2 */
  /* Virtual node exist in the aggregation tree */
  if (insertionnode ≠ NIL) then
    result := LeftSubTree (insertionnode)
    if (result ≠ NIL) then
      Insertion(result)
    result := RightSubTree (insertionnode)
    if (result ≠ NIL) then
      Insertion(result)
    DeleteNodeFromFT (insertionnode)
  /* STEP 3 */
  /* FindAncestorNode searches ancestor node */
 ancestornode := FindAncestorNode (insertionnode, defaultlevel)
  if (ancestornode ≠ NIL) then
    InsertWithAncestorNode (insertionnode, ancestornode)
  else
    InsertWithoutAncestorNode (insertionnode)
   }
```

【도 8b】

```
FindNode (prefix) {
  local node
  /* Search for node to be inserted */
  node := GetNode (prefix)
  /* Identity the insertion node */
  if (node ? NIL) then
        return (node)
  else
        return NIL
 }
```

【도 8c】

```
InsertWithoutAncestorNode (insertionnode) {
 local ancestornode, newdelegationnode, searchlevel,
       descendantnode, dynamiclevel
  /* Lemma 2-2 */
 searchlevel := defaultlevel*2 - 1
 ancestornode := FindAncestorNode (insertionnode, searchlevel)
  if (ancestornode = NIL) then
     dynamiclevel := defaultlevel
  else  /* FindDescendantNode finds node not written to FT  */
    descendantnode := FindDescendantNode (insertionnode, defaultlevel)
   /* Get adaptive level */
        if (descendantnode ? NIL) then
          dynamiclevel := GetNodePrefixLength (insertionnode)
              - GetDistPrefixLength (insertionnode, descendantnode)
        else
          dynamiclevel := defaultlevel
  newdelegationnode : = MakeDelegationNode (insertionnode, dynamiclevle)
  if (NodeNexthopVirtual (insertionnode) = TRUE) then
    InsertNodeToFT (newdelegationnode)
  else /* All inter-domain node must be inserted into FT */
    InsertNodeToFT (insertionnode)
}
```

【도 8d】

```
MakeDelegationNode (node, level) {
 local delegationnode
 /* Make a virtual delegation node and set node type */
  delegationnode := AllocateDelegationNode (node, level)
  SetNodeType (delegationnode) := DELEGATION
  SetDelegationNodeIndex (delegationnode, node)
  return (delegationnode)
}
```

【도 8e】

```
InsertWithAncestorNode (insertionnode, ancestornode) {
  local newdelegationnode, result, dynamiclevel
 dynamiclevel := SelectDelegationLevel (insertionnode, ancestornode)
  if (dynamiclevel > 0) then
     newdelegationnode := MakeDelegationNode (insertionnode, dynamiclevel);
   /* Resolve destination area */
     if (NodeNexthopVirtual (insertionnode) = TRUE) then
       InsertNodeToFT  (newdelegationnode)
     else
       InsertNodeToFT  (insertionnode)
  elseif (dynamiclevel = 0) then
    InsertNodeToFT  (insertionnode)
  else /* Check aggregatability of route */
    if (NodeNexthopVirtual  (insertionnode) = FALSE) then
          InsertNodeToFT  (insertionnode)
}
```

【도 8f】

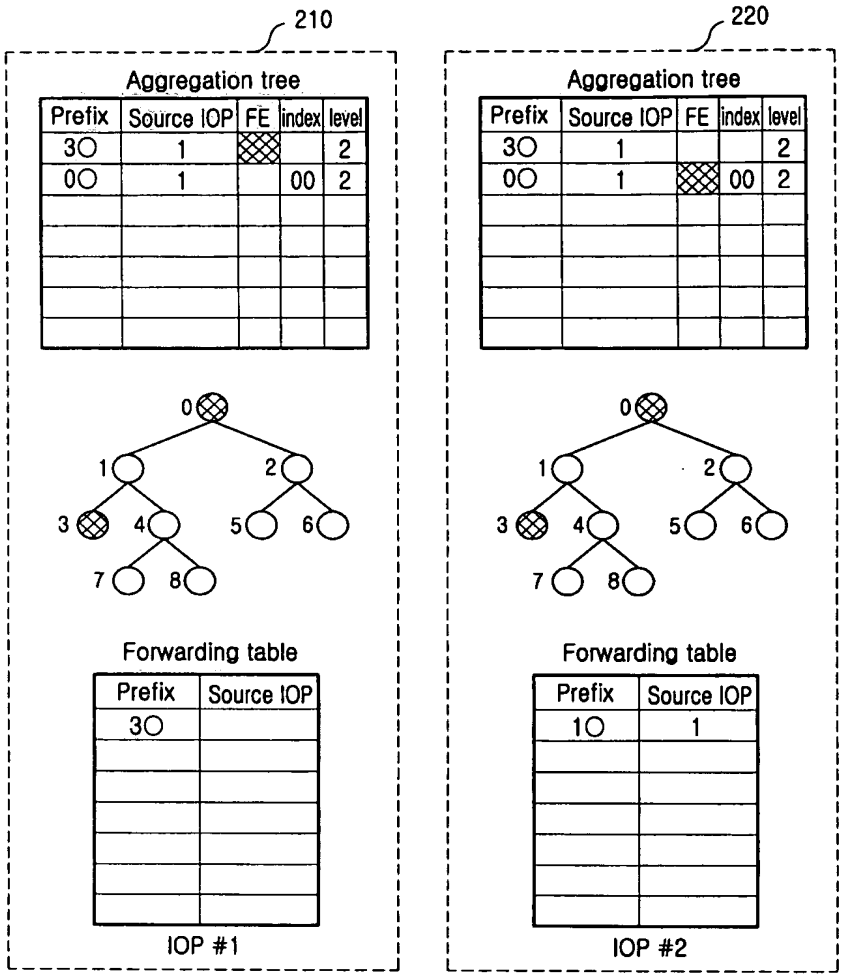
```
SelectDelegationLevel (insertionnode, ancestornode) {
  local dynamiclevel, result, searchnode, descendantnode
   descendantnode := FindDescendantNode (insertionnode, GetNodeLevel (ancestornode))
  /* Get adaptive level */
  if (descendantnode ≠  NIL) then
          dynamiclevel := GetNodePrefixLength (insertionnode)
                          - GetDistPrefixLength (insertionnode, descendantnode)
  else
    dynamiclevel := GetNodesLevel (insertionnode, ancestornode)
  /* Aggregatable case */
  if (CheckNodeUpdateToFT(ancestornode) = TRUE
    && NodeSource(insertionnode) = NodeSource(ancestornode)) then
       dynamiclevel := -1

  return dynamiclevel
}
```

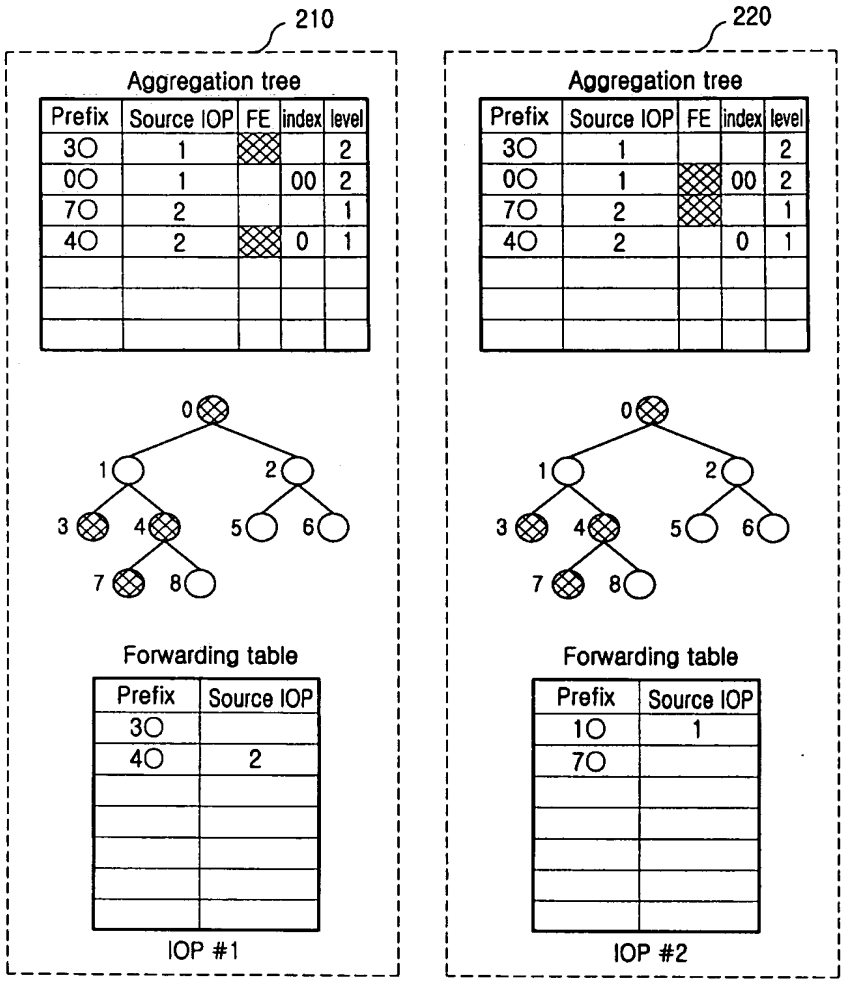
【도 8g】

```
Deletion (prefix) {
 local deletionnode, delegationnode,
       result, descendantnode
  /* STEP1 */
  deletionnode := FindNode (prefix)
  /* STEP2 */
 If (CheckNodeAggregate (deletionnode) = TRUE)
    then
    deletionnode := GetDelegationNode (deletionnode)
 If (CheckNodeUpdateToFT (deletionnode) = TRUE)
     then
    descendantnode := FindDescendantNode
                      (deletionnode, defaultlevel)
    /* Suppress deletion from FT */
    if (descendantnode ? NIL) then
    SetDelegationNode (deletionnode,
        descendantnode)
    else
      DeleteNodeFromFT (deletionnode)
  }
```

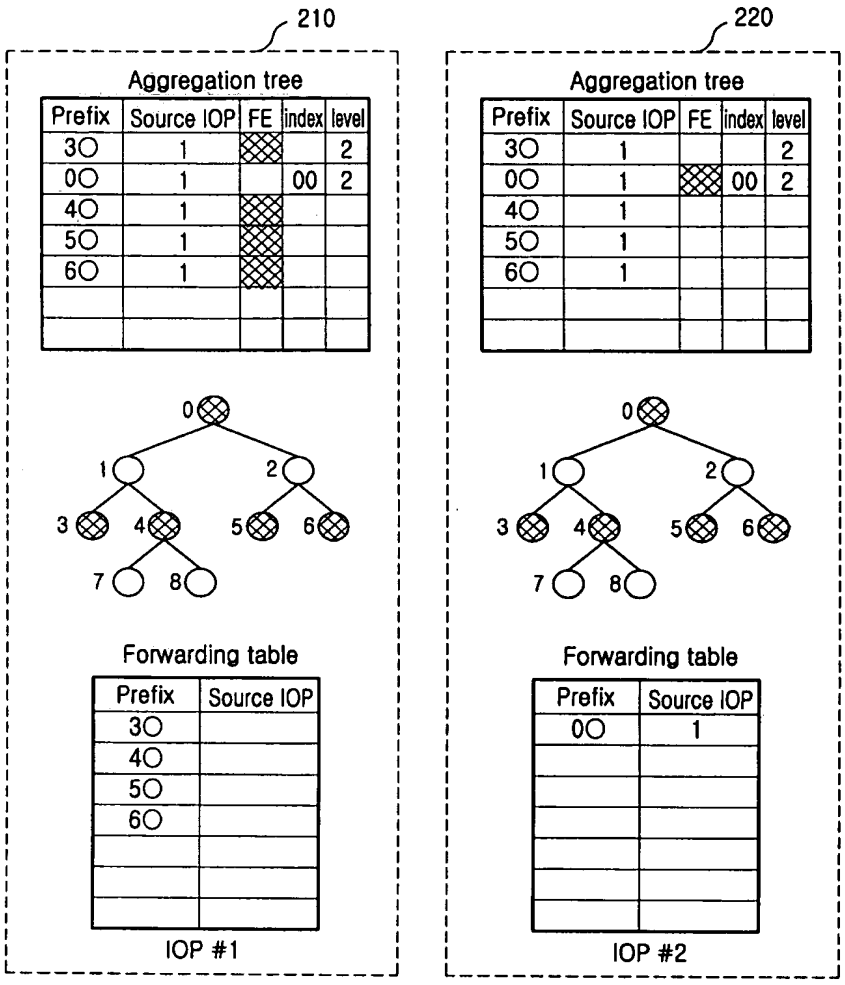
【도 9a】


210
Aggregation tree
Prefix
Source IOP
FE
index
level
3O
1
2
0O
1
00
2
0
1
2
3
4
5
6
7
8
Forwarding table
Prefix
Source IOP
3O
IOP #1
220
Aggregation tree
Prefix
Source IOP
FE
index
level
3O
1
2
0O
1
00
2
0
1
2
3
4
5
6
7
8
Forwarding table
Prefix
Source IOP
1O
1
IOP #2

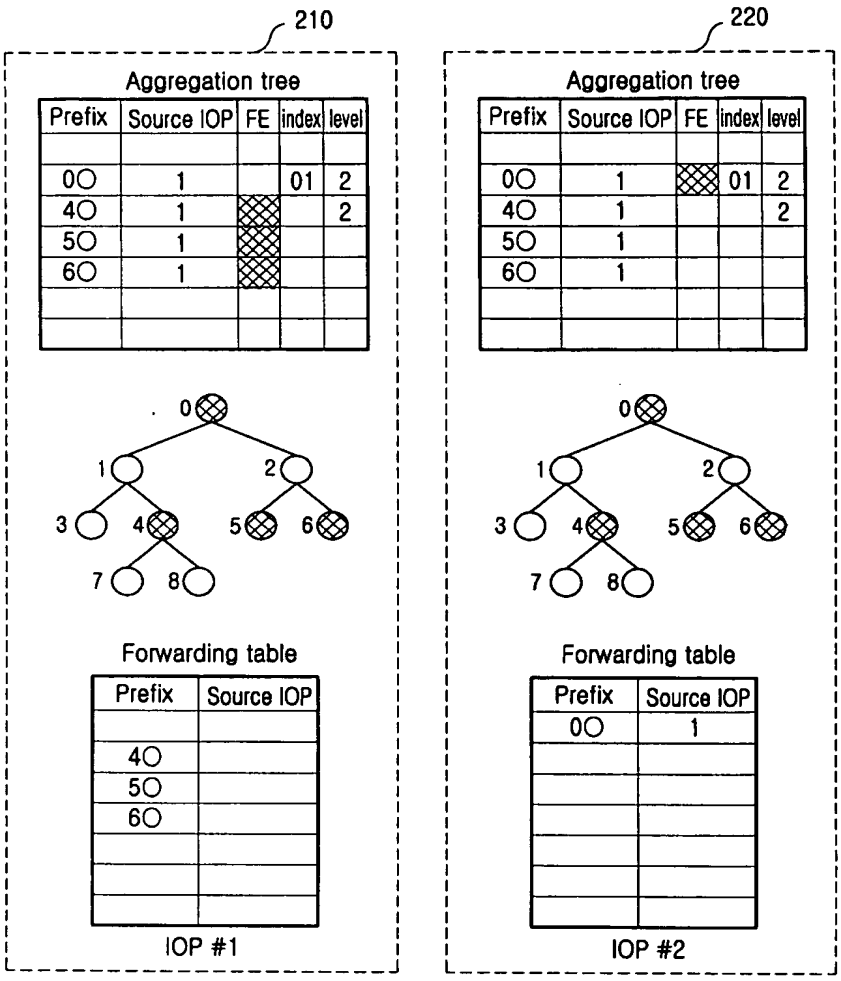
【도 9b】


210
Aggregation tree
Prefix | Source IOP | FE | index | level
3O | 1 | | | 2
0O | 1 | | 00 | 2
7O | 2 | | | 1
4O | 2 | | 0 | 1
0
1
2
3
4
5
6
7
8
Forwarding table
Prefix | Source IOP
3O |
4O | 2
IOP #1
220
Aggregation tree
Prefix | Source IOP | FE | index | level
3O | 1 | | | 2
0O | 1 | | 00 | 2
7O | 2 | | | 1
4O | 2 | | 0 | 1
0
1
2
3
4
5
6
7
8
Forwarding table
Prefix | Source IOP
1O | 1
7O |
IOP #2

【도 10a】


210
Aggregation tree
Prefix | Source IOP | FE | index | level
3O | 1 | | | 2
0O | 1 | | 00 | 2
4O | 1 | | |
5O | 1 | | |
6O | 1 | | |
0
1
2
3
4
5
6
7
8
Forwarding table
Prefix | Source IOP
3O |
4O |
5O |
6O |
IOP #1
220
Aggregation tree
Prefix | Source IOP | FE | index | level
3O | 1 | | | 2
0O | 1 | | 00 | 2
4O | 1 | | |
5O | 1 | | |
6O | 1 | | |
0
1
2
3
4
5
6
7
8
Forwarding table
Prefix | Source IOP
0O | 1
IOP #2

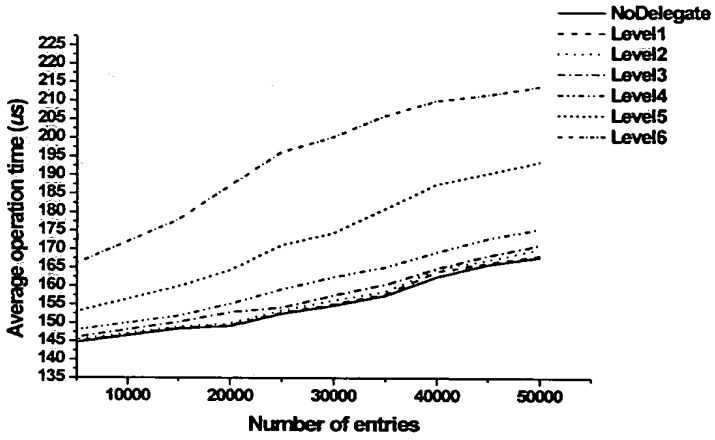
【도 10b】



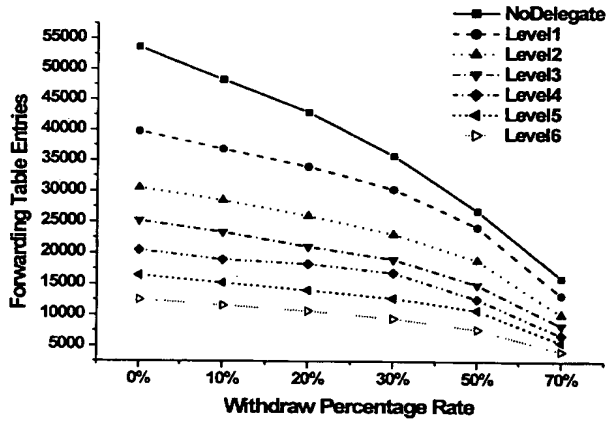
【도 11a】

Default level	Number of the entries	Number of the difference	Reduction rate
1	39,802	58	26%
2	30,574	104	43%
3	25,231	87	53%
4	20,491	118	62%
5	16,478	82	69%
6	12,554	61	77%
Original	53,632	0	0%

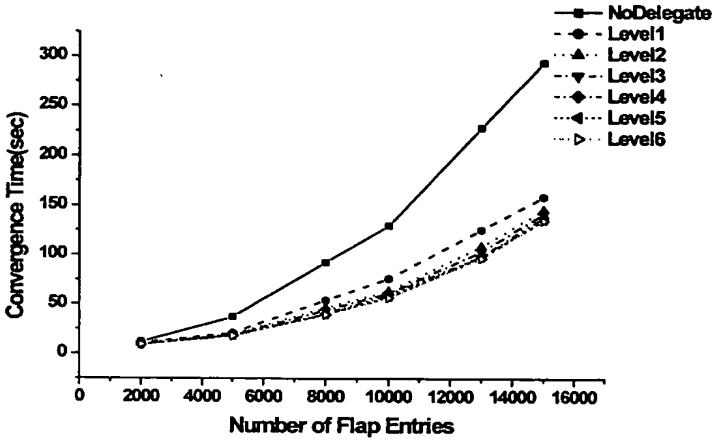
【도 11b】


NoDelegate
Level1
Level2
Level3
Level4
Level5
Level6
Average operation time (us)
225
220
215
210
205
200
195
190
185
180
175
170
165
160
155
150
145
140
135
10000
20000
30000
40000
50000
Number of entries

【도 11c】


NoDelegate
Level1
Level2
Level3
Level4
Level5
Level6
Forwarding Table Entries
55000
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0%
10%
20%
30%
50%
70%
Withdraw Percentage Rate

【도 11d】


NoDelegate
Level1
Level2
Level3
Level4
Level5
Level6
Convergence Time(sec)
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000
14000
16000
Number of Flap Entries